Case 3. Værvarsel for svaksynte

Team 11:

- Vincent Sidsore Traore (vincest)
- Alex Santos Holm (alexsho)
- Iselin Mordal Bakke (iselimb)
- Emel Kadragic (emelk)
- Mohamed Jeylani Muhuyadin (mohamejm)
- Embrik Buflod Bovollen (embrikbb)

Veiledere:

Endre Valen

Eirik Langholm

1. Presentasjon av prosjektet

1.1 Introduksjon

Gruppen vår består av fem studenter fra linjen: Informatikk: Programmering og systemarkitektur, og én student fra linjen: Informatikk: Design, bruk og interaksjon.

Rapporten gir en presentasjon av applikasjonen vår "WeatherSense", samt en gjennomgang av hvordan utviklingsprosessen har vært. Dette innebærer alt fra møter, de forskjellige sprintene og testing av applikasjonen. Rapporten starter med brukerdokumentasjon som forklarer applikasjonens funksjonalitet. Senere går vi inn på mer tekniske detaljer der vi forklarer prosessen og hvorfor vi har tatt visse valg der vi til slutt avslutter med en refleksjon rundt prosjektarbeidet.

1.2 Teamet

Alex Santos Holm (20) studerer Informatikk: programmering og systemarkitektur, og er på sitt fjerde semester. Han har i hovedsak jobbet med struktur og oppbygning av rapporten, men har også hjulpet til med både backend- og frontend programmering.

Vincent Sidsore Traore (22) studerer Informatikk: programmering og systemarkitektur, og er på sitt fjerde semester. Han har for det meste jobbet med backend-delen av appen.

Iselin Mordal Bakke (20) studerer Informatikk: programmering og systemarkitektur, og er på sitt fjerde semester. I dette prosjektet har hun fått prøvd seg mer på frontend da hun ønsket å prøve noe nytt, og har også jobbet i backend.

Mohamed Jeylani Muhuyadin (22) studerer Informatikk: design, bruk, interaksjon. Jobbet litt overalt, startet med backend og gikk videre over til design og rapportskriving.

Embrik Buflod Bovollen (21) studerer Informatikk: programmering og systemarkitektur, og er på sitt fjerde semester. Han har jobbet med både frontend (UI-design) og backend.

Emel Kadragic (21 år) går studieprogrammet Informatikk: programmering og systemarkitektur, jobbet frontend med design og layout av applikasjonen, brukertester og rapportskriving mot slutten.

1.3 Presentasjon av case

Ikke alle mennesker er født med like evner. Noen individer er født med funksjonsnedsettelser som gjør det mer utfordrende å få tilgang til og samhandle med verden rundt dem. En av disse funksjonsnedsettelsene er synssvekkelse. Dette kan gjøre det utfordrende for enkelte å holde seg informert om været. Personer med nedsatt syn kan synes det er vanskeligere å lese tekst på skjermer, tolke visuelle signaler eller se værmønstre som enkelt er synlige for andre. Dette kan føre til både følelser av isolasjon og frustrasjon, fordi de ikke klarer å holde seg oppdatert på viktig værinformasjon som kan påvirke hverdagen.

"Happiness is not something you postpone for the future; it is something you design for the present. And designing it means ensuring that every individual, regardless of their abilities, has access to the same opportunities and experiences." - Stevie Wonder

Det er derfor vi mener det er viktig å lage en værvarselapplikasjon som er spesielt utviklet for personer med nedsatt syn. Ved å gjøre det så sikrer vi at alle, uansett evne, har tilgang til nøyaktig og oppdatert værinformasjon.

Applikasjonen vår er designet for å være tilgjengelig og enkel å bruke, med funksjoner som gjør det enkelt for personer med nedsatt syn å få tilgang til informasjonen de trenger. Vi mener at alle har rett til å få tilgang til informasjon og holde seg informert.

1.4 Beskrivelse av løsning

Værmeldingsappplikasjonen er en mobilapplikasjon som er primært designet for brukere med nedsatt synsevne, der hovedfokuset vårt var å fremstille værforholdene rundt om i verden på en enkel og oversiktlig måte. Applikasjonen har tre skjermer: En skjerm for søk, en for fremstilling av værdata og en for innstillinger og informasjon. SearchScreen lar brukere søke etter værforholdene hvor som helst i verden, både ved hjelp av talekommandoer eller et tilgjengelig tastatur. MainScreen viser værmeldingen for de neste dagene, time for time for brukerens nåværende plassering med en høykontrast og “easy-to-read” design. Værskjermen gir et sammendrag på værforholdene, et detaljert varsel per time for de neste dagene, samt temperatur og vær.

2. Brukerdokumentasjon

2.1 Målgruppe

Da vi utviklet værvarselapplikasjonen ønsket vi å sørge for at den kom til å møte behovene til så mange brukere som mulig. Været påvirker alle uavhengig av alder, sted eller bakgrunn. Dette er derfor vi ønsket å lage en applikasjon som ville være tilgjengelig og enkel å bruke for en universel målgruppe. Eldre personer rundt 40 år, som er mer utsatt for

synssvekkelse og forstår mindre av dagens teknologiske standarder, kan ha ulike behov og preferanser når det kommer til teknologi. De foretrekker kanskje et enklere grensesnitt eller større skriftstørrelser som gjør applikasjonen lettere å lese. Det finnes også forskjellige typer synssvekkelser og vi har derfor designet applikasjonen for de vanligste synssvekkelsene: Cataract, Glaucoma og macular degeneration. Dette er grunnen til at vi sørget for at applikasjonen vår er designet for å være brukervennlig og inkluderende, med funksjoner som gjør det enkelt for brukere i alle aldre å få tilgang til den informasjonen de trenger.

2.2 Funksjonalitet

Formålet med applikasjonen vår er å gi en værmelding til brukere med nedsatt synsevne, på en nøyaktig og lett tilgjengelig måte, der værinformasjon fremstilles i et inkluderende samt brukervennlig format. Vi har implementert en rekke funksjonaliteter som skal hjelpe med visjonen. Vi har prøvd å holde applikasjonen kompakt og lett oversiktlig for de synssvekkede, slik at det skal være like lett for dem å bruke applikasjonen. Brukerdokumentasjonen skal hjelpe deg med å få en dypere forståelse rundt applikasjonen og dens funksjonaliteter.

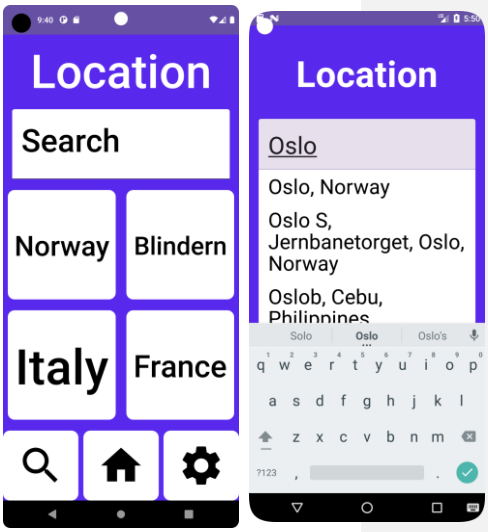
2.2.1 Forside med vær

Når du starter applikasjonen for aller første gang, vil du bli bedt om å dele lokasjonen din. Dette gjøres for enklere og raskere visning av området brukeren befinner seg i. MainScreen er delt inn i tre hoveddeler. Den øverste delen fremstiller de valgbare dagene fremover og gjennomsnittstemperaturen for de dagene. Delen på midten av skjermen viser været time for time for den valgte dagen ved å swipe til høyre på kortet. Den nederste delen av skjermen er en navigasjonsbar som bytter til de forskjellige skjermene. Skjermene vi har implementert er SearchScreen til venstre, MainScreen i midten og SettingScreen til høyre. På toppen av MainScreen er det sentralt plassert en stjerne som er en knapp der bruker kan legge til den oppgitte lokasjonen som en favoritt-lokasjon. Når denne knappen trykkes, blir lokasjonen lagret i en liste som representeres i SearchScreen.

2.2.2 Text-to-speech

MainScreen har også text-to-speech på hvert eneste kort. Dette betyr at om bruker trykker på for eksempel det midterste kortet på MainScreen vil det leses opp alt som står der fra øverst til venstre, altså lokasjonen, tiden, hvordan været er så temperaturen. Text-to-speech vil fortelle at du kan sveipe mot høyre for å se de neste timene for den valgte dagen. Bruker kan trykke på de forskjellige dagene og da vil det leses opp hvilken av dagene som er blitt valgt, etterfulgt av gjennomsnittstemperaturen for den valgte dagen. Text-to-speech er og implementert på den måten at de forskjellige valgene på navigasjonsbaren, helt nederst på skjermen, vil bli lest opp ved trykk. Inne på MainScreen vil det leses opp at skjermen blir lastet inn på nytt ved trykk av denne knapp. Text-to-speech vil også kunne lese opp ved behandling av favorittlokasjoner.

2.2.3 Søkefelt



SearchScreen fungerer som et verktøy for å finne og få tilgang til værinformasjon fra ulike steder i verden. Med et brukervennlig grensesnitt tilbyr denne skjermen et søkefelt som lar brukeren skrive inn et ønsket stedsnavn. Videre hentes nøyaktige værdata fra den gitte lokasjonen. Etter å ha trykket på en ønsket lokasjon, blir brukeren sendt tilbake til MainScreen som viser været for det gitte stedsnavnet. I tillegg er SearchScreen utstyrt med praktiske funksjoner som inneholder fremstilling for lagrede favorittsteder, som gjør at brukerne enkelt kan hente værinformasjon fra sine favorittsteder. For å gjøre det lettere for brukere med nedsatt synsevne, er det mulig å bruke den innebygde mikrofonen på telefonens tastatur, slik at brukere kan bruke talekommandoer for å søke etter værforhold. I likhet med MainScreen, har SearchScreen også text-to-speech på navigasjonsbaren.

2.2.4 SettingScreen & MainScreen

SettingScreen er en av skjermene til applikasjonen som fremstiller informasjon om selve applikasjonen. Dette innebærer de brukte ikonene og API-ene som er brukt. Vi hadde opprinnelig planer om å implementere funksjonalitet som ville tillate brukerne å selv velge mellom forskjellige fargekontraster i applikasjonen, etter hensyn til forskjellige typer fargeblindheter. Dessverre ble denne funksjonaliteten ikke gjennomført, da vi valgte å prioritere funksjonaliteten på de andre skjermene og deres funksjoner grunnet tidsmessige årsaker.

Selv om denne spesifikke funksjonaliteten for å tilpasse kontraster ikke ble fullført, har applikasjonen likevel høye kontrastfarger mellom svart, hvitt og lilla. Dette er et tiltak som forbedrer lesbarhet og tilgjengelighet for mange brukere, inkludert de med fargeblindhet og nedsatt synsevne. Kontrastforholdet mellom lillafargen og hvitfargen er målt til 7.12:1, og mellom hvit og svart er kontrastforholdet målt til 21:1. Dette nivået av kontrast oppfyller kravene i WCAG 2.1 nivå AA for god lesbarhet.

Den valgte fargekombinasjonen sikrer dermed tydelighet, synlighet og lesbarhet for brukere av applikasjonen. Dette bidrar til å sikre en god brukeropplevelse for et bredt spekter av brukere, inkludert de med forskjellige fargeblindheter og nedsatt synsevne.

2.3 Kjørbarhet (Hvilke plattformer applikasjonen kan kjøre på)

Applikasjonen “WeatherSense” kan kjøres på Android-smarttelefoner og nettbrett. Under utvikling av applikasjonen, har vi brukt Android Studios sin emuleringsfunksjon for å emulere og fremstille applikasjonen. Hovedsakelig ble Pixel 5 og Pixel 4 med API-nivåene 24 og 30 brukt for å verifisere kvalitet. Hvis text-to-speech skal fungere, trenger Android-

smarttelefonene å ha Google Play-applikasjonen installert slik at den har en TTS-engine tilkoblet.

2.3.1 Hvordan aksessere applikasjonen

Applikasjonen er ikke tilgjengelig for offentlig nedlasting på Google Play, men den kan lastes ned via GitHub. Vi har opprettet en privat GitHub-side der applikasjonen kan lastes ned og brukes. For å få tilgang til siden, skal man kontakte en av utviklerne direkte.

For å teste applikasjonen, er man nødt til å laste ned prosjektet for så videre åpne det i Android Studio. Brukeren må selv sette opp en emulator med Google Play eller koble til en fysisk Android-enhet, slik at applikasjonen kan bli vist på enten emulatoren i Android Studio eller på en fysisk Android-enhet. Etter å ha gjort klar en enhet med TTS stilt inn, trykker brukeren på den grønne trekanten i verktøylinjen øverst på Android Studio eller holder inne shift+f10 på tastaturet. På emulatoren må brukeren forsikre seg om at dato og klokkeslett stemmer, hvis ikke kommer applikasjonen til å krasje.

3. Kravspesifikasjoner og modellering

3.1 Funksjonelle krav

Tidlig i prosjektet så definerte vi våre funksjonelle og ikke-funksjonelle krav for applikasjonen. Senere valgte vi å legge til, samt fjerne, flere funksjonelle krav basert på behov og gjenværende tid. Disse kravene som ble valgt var basert på hva vi mente kom til å være verdifulle for den gitte målgruppen, og hjalp oss å sette et grunnlag for startfasen. Kravene er sortert etter hva vi mente var viktigst, noe som gjorde prioritering av arbeidsoppgaver enklere. Kravene ble basert på allerede eksisterende empirisk forskning der WCAG ble brukt som en retningslinje for hvordan man oppretter en universell utformet applikasjon.

De viktigste funksjonelle kravene vi til slutt endte med å velge var følgende:

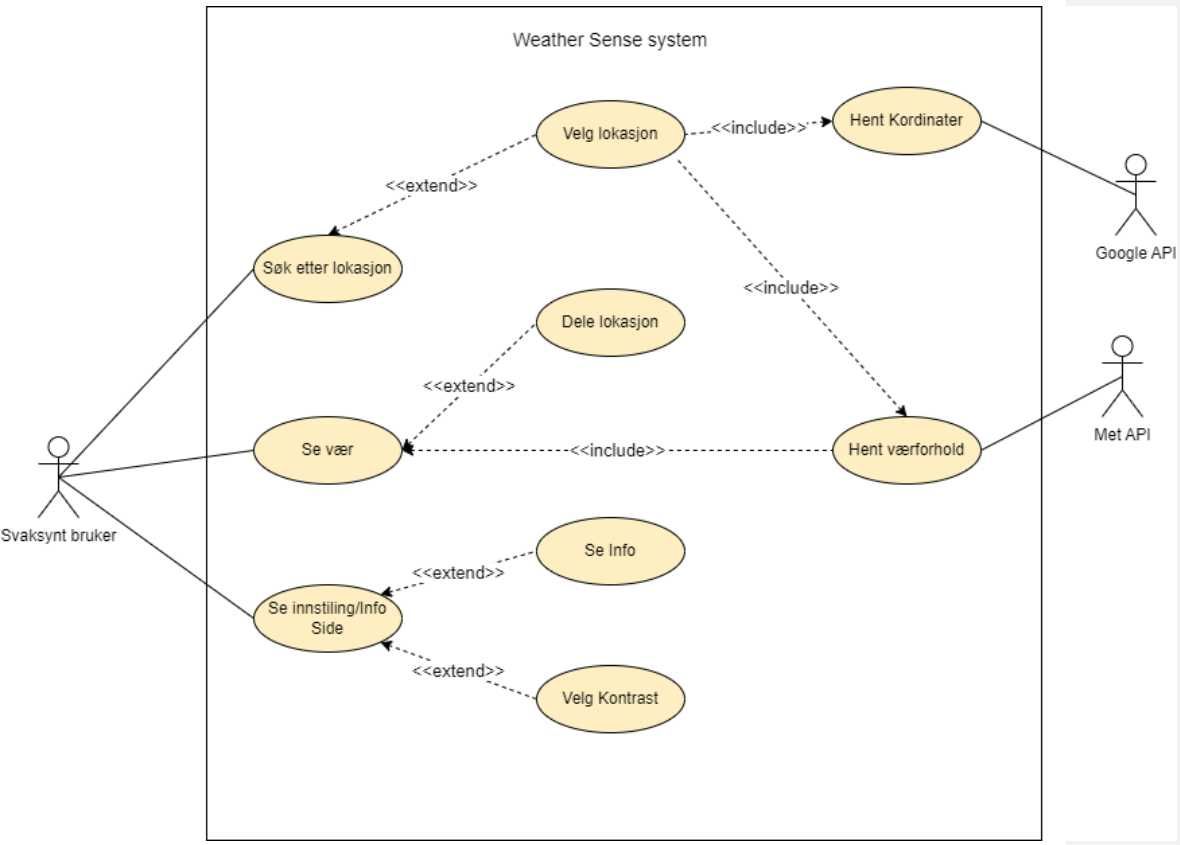
- Applikasjonen skal vise værmelding for hver time for angitt dag

- Applikasjonen skal representere data og annet med store elementer, tilpasset svaksynte(knapper, tekst, figurer m.m.)
- Applikasjonen skal ha en søkelinje for å velge lokasjon
- Applikasjonen skal ha tekst til tale-funksjonalitet

3.2 Use Case

For å tydeliggjøre de ulike funksjonalitetene i applikasjonen, utarbeidet vi et use case diagram (figur 1). Dette diagrammet ga oss en oversikt over hvordan systemet er strukturert.

Use case-diagram for WeatherSense



3.3 Tekstlige beskrivelser

De følgende tekstlige beskrivelsene beskriver de viktigste funksjonelle kravene til applikasjonen.

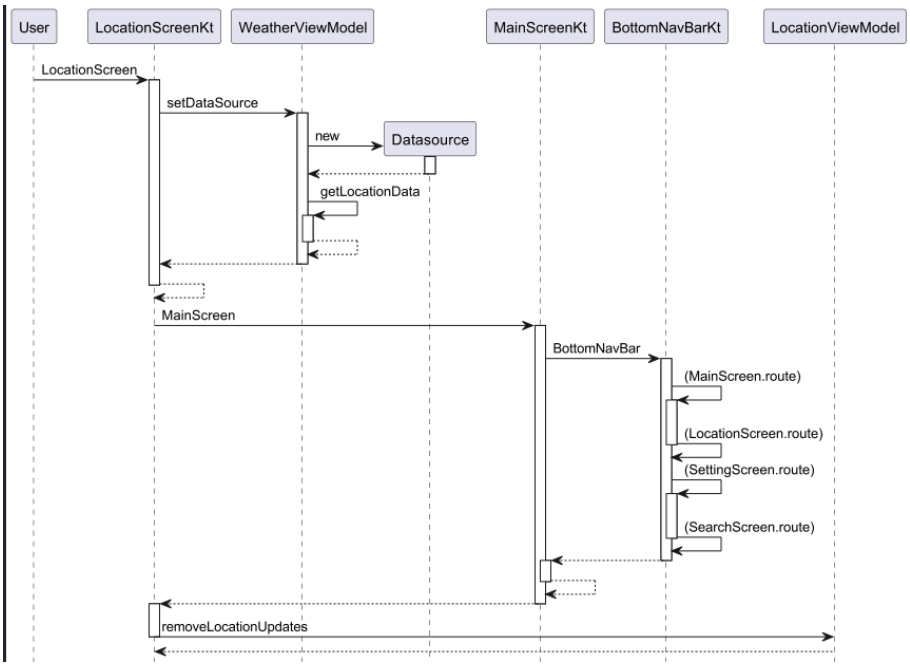
Navn	**Åpne app og se været for din lokasjon**

Primæraktør	Bruker
Sekundæraktør	Google, MET
Prebetingelse	Applikasjonen har tilgang til internett, Applikasjonen er ennå ikke åpnet.
Postbetingelse	Applikasjonen viser været for brukerens lokasjon
Hovedflyt	1. Bruker åpner applikasjonene 2. Applikasjonen viser en loading screen 3. Applikasjonen spør om lokasjonen til bruker 4. Applikasjonen tar koordinatene bruker og sender det til Location Forecast api som henter værdata 5. Applikasjonen fremstiller værdata for lokasjonen til bruker
Alternativ flyt	3.1 Bruker sier nei til deling av nåværende lokasjon 3.2 Lokasjonen blir satt til Oslo 5.1 Bruker trykker på værdata kortet 5.2 Text-to-speech leser opp lokasjonen, tiden, været og temperaturen til brukeren

Navn	**Søk på lokasjonen Bergen og fremstill været i Bergen**
Primæraktør	Bruker
Sekundæraktør	Google, MET
Prebetingelse	Applikasjonen viser SearchScreen
Postbetingelse	Fremstiller værdata til Bergen

Hovedflyt	1. Bruker skriver inn “b” i søkefeltet 2. Applikasjonen viser forslag til lokasjon som starter på b 3. Bruker skriver resten av ordet “ergen” 4. Bruker trykker på alternativet Bergen 5. Applikasjonen henter koordinatene til Bergen og sender det videre til Location Forecast APIet og henter korresponderende værdata 6. Applikasjonen sender bruker til MainScreen og fremstiller værdata til Bergen
Alternativ flyt	1.1 Bruker benytter seg av innebygd mikrofon og sier Bergen 1.2 Bruker trykker på alternativet Bergen 1.3 Applikasjonen henter koordinatene til Bergen og sender det videre til Location Forecast APIet og henter korresponderende værdata 1.4 Applikasjonen sender bruker til MainScreen og viser værdata til Bergen

3.4 Sekvensdiagram



SekvensDiagrammet tar oss gjennom prosessen når brukeren åpner appen. Det starter fra LocationScreen og fører oss frem til Mainscreenen. Deretter blir removeLocationUpdates kalt. Denne er ikke implementert i appen, da vi ikke fikk tid til dette.

3.5 Klassediagram

Klasse Diagrammet er delt opp i 3 uavhengige deler. Det oransje viser til hva vi henter fra location forecast. Det grønne viser til dataklassene vi oppbevarer, sharedpreferences og brukeren sin nåværende lokasjon. Og til slutt er de blå dataklassene som oppbevarer informasjonen vi får fra google places Api.

3.6 Objektorienterte prinsipper

I vårt program har vi benyttet oss av objektorientert programmering da det finnes flere fordeler med dette. De objektorienterte prinsippene vi har tatt i bruk gjennom prosjektperioden er innkapsling, arv, polymorfi og abstraksjon. Disse prinsippene bidrar til å gjøre programmet mer lesbart, vedlikeholdbart og gjenbrukbart. Applikasjonen er bygget opp ved bruk av MVVM-modellen, som i grunnlag benytter objektorientert programmering for å separere "concerns" og gjenbruk av kode. Våre views (screens) kaller på metoder fra ViewModel, som oppdaterer Model (UiState). Vår applikasjon består av ViewModels og Models representert som klasser, samt View består av "composables"-funksjoner, som er designet rundt funksjonell programmering, men tar nytte av ViewModels og modellen. Dette gjør at vi fortsatt følger prinsippet om innkapsling og polyformisme. View-komponentene representerer UI-komponenter som kan gjenbrukes. Dette er et kjerneprinsipp i objektorientert programmering.

Innkapsling

I vårt program har vi blant annet implementert FavoriteLocation-klassen, som er en del av Model-komponenten i MVVM-arkitekturen. Denne klassen har en offentlig metode som heter saveFavoriteLocation(). Denne metoden tillater interaksjon med klassens interne struktur uten å avsløre alle detaljene. Spesifikt benytter saveFavoriteLocation() seg av en privat variabel, sharedPreferences, for å lagre data om favorittlokasjoner.

Selv om sharedPreferences er skjult for resten av systemet, gjør saveFavoriteLocation()-metoden det mulig for andre deler av systemet å lagre favorittlokasjoner. Dette er en praktisk anvendelse av innkapslingsprinsippet i objektorientert programmering, der dataene er skjult, men tilgjengelige gjennom offentlige metoder.

LocationViewModel-klassen, en del av ViewModel-komponenten i MVVM-arkitekturen inneholder en instans av FavoriteLocation klassen. Dermed kan LocationViewModel-klassen bruke saveFavoriteLocation()-metoden for å lagre favorittlokasjoner på vegne av View-komponenten. Dette illustrerer også innkapsling, da View-komponenten kan lagre en favorittlokasjon uten å behøve detaljert kunnskap om hvordan FavoriteLocation håndterer lagringen.

Abstraksjon

Et eksempel på abstraksjon er LocationUiState-klassen, som er en "sealed class" med underklassene "Loading", "Error" og "Success".Denne klassen brukes til å abstrahere forskjellige tilstander UI-et kan være i når det gjelder visning av lokasjonsdata. Det kan enten være i en lastetilstand (representert av Loading), en suksess-tilstand der det har mottatt data (representert av Success), eller en feiltilstand (representert av Error).

Når vi håndterer en LocationUiState i programmet vårt, trenger vi ikke å bekymre oss for spesifikke detaljer om hva som skjer i bakgrunnen når disse tilstandene oppstår. Vi behøver kun å sjekke hvilken underklasse av LocationUiState vi har, og håndtere det tilsvarende. Dette er kjernen i abstraksjon - å skjule detaljerte implementasjoner og utsette bare det som er nødvendig for bruker.

Arv

Når det kommer til arv, har vi for eksempel LocationViewModel-klassen som arver fra Android sin AndroidViewModel-klasse, som igjen arver fra Android sin ViewModel-klasse. Ved at LocationViewModel-klassen arver fra AndroidViewModel-klassen, kan klassen arve egenskaper fra både AndroidViewModel-klassen og ViewModel-klassen. Vår LocationViewModel bruker for eksempel viewModelScopet CorutineScope, og OnCleared()-funksjonen som stammer fra ViewModel-klassen.

Polymorfi

OnStatusChanged()- funksjonen, som vi benytter oss av inne i LocationViewModel-klassen, er et eksempel på polymorfi. Ved å bruke "override" på denne funksjonen, gir vi vår egen

implementasjon av funksjonen fra LocationListener-grensesnittet. Vi oppretter et nytt objekt av LocationListener og tilpasser metoden, som fører til at vi kan behandle forskjellige implementasjoner av LocationListener, på en enhetlig måte. Dette gir oss gjenbrukbart og vedlikeholdbart program.

4. Produktdokumentasjon

4.1 Teknologier

Android studio

Beslutningen om å bruke Android Studio for prosjektet ble drevet av en kombinasjon rundt tidligere erfaringer, samt at det var et krav til prosjektet vårt. Android Studio er det offisielle integrerte utviklingsmiljøet (IDE) for Android-apputvikling. Android Studio tilbyr en mengde viktige verktøy og ressurser som for eksempel emulering av Android-enheter for testing. Dette gjorde sånn at vi enkelt kunne teste applikasjonen på måten den skulle fremstilles.

Jetpack Compose

Teamet vårt bestemte seg for å bruke Jetpack Compose for prosjektet på grunn av dets brukervennlighet og det faktumet at alle i teamet hadde tidligere erfaringer med det. Jetpack Compose, som er et moderne UI-verktøysett for utvikling av Android-applikasjoner, gir en enkel og intuitiv måte å lage UI-komponenter. Vi fant tidlig ut at læringskurven for Jetpack Compose var minimal, slik at vi raskt kunne forstå konseptene og begynne å bygge brukergrensesnittet og skjermer effektivt. Fordi teammedlemmene allerede hadde brukt Android-Studio som IDE og Jetpack Compose, som er sømløst integrert med Android Studio, så ga det mer mening å bruke dem sammen.

Ktor

For HTTP-bibliotek, valgte vi å bruke Ktor til prosjektet fordi nesten alle i teamet hadde tidligere erfaringer med det fra Oblig 2. Denne kjennskapen til Ktor tillot oss å utnytte vår eksisterende kunnskap og ekspertise, noe som førte til en raskere utviklingsprosess med færre problemer. Siden KTOR oppfylte alle våres HTTP-relaterte krav, trengte ikke teamet å bruke noe annet HTTP-bibliotek. Teamet fant KTOR som et pålitelig valg for prosjektet, basert på våres tidligere positive erfaringer med det i tidligere obligatorisk innlevering.

Gson

Gson ble brukt for deserialisere JSON til Kotlin-objekter fra API-responsene. I likhet med KTOR, ble gson benyttet under Oblig 2 og var hovedgrunnen til at vi fortsatte med bruken av gson under prosjektet.

4.2 Systemet/Systemflyt

MainActivity

Det første som skjer når applikasjonen startes, er et kall til systemet som sjekker om bruker har tillatt posisjonsdeling. Så starter Navigation App Host som igjen starter View Modellene som kommer til å måtte sende inn til Screens. Nå kan NavHosten sende bruker til start destinasjonen som blir LocationScreen.

LocationScreen

LocationScreen blir brukt som et mellomledd slik at applikasjonen kan skille mellom det å hente nåværende lokasjon og en søkt opp lokasjon. Siden det kun blir kalt ved oppstart og hvis bruker i MainScreen trykker på den midtre knapp i navigasjonsbaren, som da blir en knapp for rekomposering. Den observerer locationUiNewState fra LocationViewModel ved hjelp av collectAsState(). Når locationUiNewState endres, blir LocationScreen komponert på nytt for å reflektere den nye tilstanden. Når locationUiNewState er LocationUiState.Loading, viser LocationScreen en indikator for lasting. Dette er den innledende tilstanden og også tilstanden under prosessen med å hente lokasjon. Når locationUiNewState er LocationUiState.Success, setter LocationScreen lokasjonsdata som

datakilde for WeatherViewModel og setter whichMain-variablen til true. Fordi locationUiNewState blir observert ved hjelp av collectAsState(), vil denne blokken bli utført så snart locationUiNewState endres til LocationUiState.Success, uten å blokkere hovedtråden. Hvis whichMain er true, navigerer LocationScreen til MainScreen.

LocationViewModel

LocationViewModel er en Android ViewModel som er ansvarlig for å hente lokasjonsdata og gi den til brukergrensesnittet. Når LocationViewModel blir opprettet, kaller den getCurrentLocation() i sin init blokk. getCurrentLocation() sjekker først om applikasjonen har de nødvendige tillatelsene for å få tilgang til lokasjonsdata. Hvis ikke, setter den en standard lokasjonsverdi (Oslo) og returnerer. Hvis applikasjonen har tillatelse til å få tilgang til lokasjonsdata, ber getCurrentLocation() om lokasjonsoppdateringer fra LocationManager. Når lokasjonen endres, blir onLocationChanged() kalt. Denne metoden starter en Corutine i ViewModel's scope, som setter de nye lokasjonsdataene i _locationUiNewState, og endrer tilstanden til LocationUiState.Success. Denne operasjonen er asynkron og blokkerer ikke hovedtråden.

Ellers har LocationViewModel funksjoner som jobber sammen med FavoriteLocation-objektet for å hente, lagre og sjekke brukerens favoriserte lokasjoner. getLocationName() går også under viewModellen men på grunn av store kostnader i Apiet, returnerer den kun “Current Location”. Ellers ville denne kalt Google sitt geocode API for å hente lokasjonsnavn.

MainScreen

MainScreen er hovedskjermen i applikasjonen, og bruker flere komponenter for å presentere data i forskjellige formater. Først samles data fra WeatherViewModel til en weatherUiState-variabel som brukes for å skape et responsivt grensesnitt. Skjermkonfigurasjonen innhentes og brukes til å beregne størrelsen for ulike elementer som skal vises på skjermen. En liste av datoer blir opprettet for å representere dagens dato og de neste ni dagene, noe som blir brukt for å vise værdata for fremtidige dager.

Deretter, avhengig av weatherUiState:

1.Når data lastes (Loading), vises en indikator for brukeren.

2.Ved suksess (Success) vises værdata gjennom bruk av Scaffold og andre komponenter. Dette inkluderer en liste med fremtidig værdata, og værdata for forskjellige tidspunkter for den valgte dagen. WeatherCardDays , WeatherCard og BottomNavBar nederst.

3.Ved en feil (Error) vises en feilmelding på skjermen.

WeatherCard og WeatherCardDays er da funksjoner som presenterer værdata på en strukturert og visuell måte. WeatherCard tar inn flere parametere, inkludert bredde og høyde for kortet, høyde for ulike elementer på kortet, Ui-tilstanden, lokasjonen, indeksen for det aktuelle tidsintervallet, og en ViewModel for text-to-speech funksjonalitet. Denne funksjonen bruker data fra Ui-tilstanden til å vise detaljer som tidspunkt, temperatur og vær-symbol for det aktuelle tidsintervallet. Når brukeren klikker på kortet, genereres en streng med relevant informasjon som leses opp ved bruk av tekst-to-speech funksjonaliteten. WeatherCardDays er en mindre versjon av WeatherCard som viser værdata for en hel dag, inkludert gjennomsnittstemperaturen. Ved å telle opp temperaturen for dagen og så dele på antall timer som er igjen. Dette skaper en interaktiv og brukervennlig applikasjon som viser værdata på en strukturert og forståelig måte. En annen funksjon MainScreen har er å kunne lagre gjeldende lokasjon som favoritt, ved hjelp av LocationViewModel sin kobling til FavoriteLocation.

FavoriteLocation

FavoriteLocation er en klasse som håndterer lagring og henting av favorittlokasjoner i en Android-applikasjon. Den bruker SharedPreferences for å lagre data på enheten, noe som gjør at dataene overlever app-omstarter og er tilgjengelig selv når applikasjonen ikke kjører. Klassen tar inn en Context i konstruktøren, som er nødvendig for å få tilgang til SharedPreferences. Her initialiseres SharedPreferences med navnet "favorite_locations", og de lagres i private modus (Context.MODE PRIVATE), som betyr at kun denne applikasjonen har tilgang til disse dataene. saveFavoriteLocation-metoden lagrer en favoritt lokasjon, inkludert navnet på stedet, breddegraden, lengdegraden, og en boolsk verdi som indikerer om stedet er en favoritt. Den bruker en roterende indeks (fra 0 til 3) for å

begrense antall lagrede steder til fire. Hver gang en ny lokasjon lagres, overskrives den eldste lokasjonen. Så sjekker og henter de andre 2 metodene ut dataen.

Weather ViewModel

WeatherViewModel er en ViewModel-klasse i Android-utvikling som brukes for å håndtere data som er nødvendig for UI. Denne klassen er ansvarlig for å hente værdata basert på bredde- og lengdegrad. Når WeatherViewModel instansieres, tar den inn bredde- og lengdegrad som argumenter. Disse brukes til å initialisere dataSource, som er en instans av klassen Datasource. Denne klassen er ansvarlig for å utføre selve nettverk forespørselen og returnere dataene. I init-blokken kaller WeatherViewModel på metoden getLocationData(). Denne metoden starter en Coroutine som utfører nettverk forespørselen på en bakgrunns tråd (Dispatchers.IO). Den prøver å hente data fra dataSource, og oppdaterer _weatherUiState basert på resultatet.

DataSource

Datasource-klassen bruker KTOR, et Kotlin-bibliotek for å utføre nettverk forespørsler for å hente data fra en gitt URL. Den bruker også Gson-biblioteket til å konvertere den hentede JSON-responsen til et Location Data-objekt. Hvis det oppstår en feil under denne prosessen, logges feilen og kastes videre slik at den kan fanges og håndteres i WeatherViewModel.

BottomNavBar()

BottomNavBar er designet for å vises i bunnen av applikasjonens skjerm og gir brukeren mulighet til å navigere mellom forskjellige skjermer i applikasjonen. Avhengig av den boolske verdien av refreshingMain-parameteret, vil den midterste knappen enten navigere brukeren til MainScreen eller LocationScreen. Hvis refreshingMain er satt til true, navigerer knappen brukeren til LocationScreen. Hvis den er satt til false, navigerer den til MainScreen.

Til slutt inneholder hver knapp en 'Image' komponent som viser et ikon. Disse ikonene representerer søk, vær og innstillinger, og er ressurser som er lagret i applikasjonen.

SearchScreen()

SearchScreen-funksjonen tar inn flere argumenter, inkludert navController , en lambda funksjon for håndtering av søke forespørsler, en LocationViewModel og TextToSpeechViewModel. Funksjonen oppretter deretter en serie med variabler som styrer skjermens utseende og oppførsel. Disse inkluderer variabler for å håndtere tekstfeltet som brukes til søk, en liste over forslag basert på det brukeren skriver inn, og en liste over favorittsteder. Etter dette bruker SearchScreen-funksjonen LaunchedEffect-funksjonen for å sende en forespørsel til Googles Places API for å hente stedsforslag basert på det brukeren har skrevet inn i søkefeltet. Disse inkluderer et tekstfelt for søk, en liste for å vise søkeforslag, og en serie med knapper for favorittsteder. Videre tar SavedLocations-funksjonen av seg visningen av favorittsteder. Den tar inn en liste med favorittsteder og håndterer klikk på disse for å utføre et søk basert på det valgte stedet. LocationSuggestion-funksjonen håndterer visning og funksjonalitet for de individuelle søke forslagene. Når et forslag klikkes på, henter den koordinatene for det valgte stedet fra Google Places APIet og sender dem videre til søkefunksjonen. Til slutt brukes fetchCoordinates-funksjonen til å hente koordinater for et gitt sted fra Google Places APIet. Den sender en forespørsel til APIet og behandler svaret for å trekke ut og returnere de relevante koordinatene.

SettingScreen()

SettingScreen er en composable funksjon som viser innstillingsmenyen for applikasjonen. Den bruker en Scaffold som basisstruktur som inneholder bottomnavbaren vår og innhold. Funksjonen tar i bruk to argumenter: navController og textToSpeechViewModel. Høyden og bredden på skjermen hentes fra konfigurasjonen. Innholdet i SettingScreen er en kolonne med forskjellige komponenter, inkludert en tittel og et ExpandableCard.

ExpandableCard er en annen composable funksjon som skaper et utvidbart kort. Disse kortene bruker vi for å fremstille ulike deler av innstillingene, som informasjon om applikasjonen, kontrastvalg, informasjon om ikonene, og hvilke APIer som brukes. Hver av disse seksjonene kan utvides eller trekkes sammen ved å trykke på dem, basert på boolske verdier.

Når en bruker trykker på en av kortene, blir textToSpeechViewModel brukt for å endre tekstfeltet sin verdi til tittelen på det klikkede kortet og deretter utløse text-to-speech-funksjonaliteten. Dette gir brukere med synshemninger en bedre forståelse av hvilken del av innstillingene de interagerer med.

4.3 Design

Designet spilte en betydelig rolle i utviklingen av applikasjonen for svaksynte. Med hensyn til brukernes behov og utfordringer, var det viktig å skape et grensesnitt som var intuitivt, tilgjengelig og enkelt å bruke. I løpet av designprosessen tok vi mange viktige beslutninger og utforsket flere alternativer.

Vi startet med å designe ulike prototyper i Figma og på papir. Dette ga oss muligheten til å visualisere og eksperimentere med forskjellige designløsninger.Vi la stor vekt på å forstå hvordan svaksynte brukere ville kunne samhandle med grensesnittet og hvilke funksjonaliteter som ville være mest effektive for deres behov. For å sikre at designet oppfylte forventningene og behovene til de svaksynte brukerne, gjennomførte vi også brukertester.

Farger

Farger var et viktig element for applikasjonen. Siden applikasjonens målgruppe og primæraktører er svaksynte, valgte vi at hovedfargene ville være en dyp nyanse av lilla og hvit. Dette skaper en fin kontrast i applikasjonen, samt i tekst og symboler. Dette gjør det enklere å se informasjonen på skjermen. I tillegg ble svart farge brukt på skriften for å vise informasjon på applikasjonen fra sted, klokkeslett og temperatur, til og med svart farge på symboler. Dette gjør det lett å skille mellom informasjon. All tekst er over hvit bakgrunn som gjør det lett for brukeren å se viktig informasjon.

Skrifttype/Størrelse

Vi ønsket at teksten skulle være leselig og tydelig for brukerne våre, men vi mente også at brukeren selv skal bestemme teksttypen deres. Derfor valgte vi FontFamily.Default som betyr at fonten på applikasjonen vil være det samme som det som er valgt på enheten til brukeren. Dette gjør sånn at uansett hva så vil det være en font som brukeren selv er vant med og ikke noe nytt for dem. Så hvis det ikke er endret på innstillingene på Android-enheten, vil vanligvis enhetens standard font være Roboto.

Ikoner

I applikasjonen vår har vi valgt å bruke flere gjenkjennbare ikoner. Dette gjør at brukere lettere kan forstå funksjonaliteten bak knappene. Forstørrelsesglass representerer der man kan søke etter lokasjoner, altså SearchScreen. Hus-ikonet er for å vise hvordan brukeren navigerer til MainScreen som fremstiller været til brukeren og tannhjulet er for å vise til bruker hvor SettingScreen er.

Navigasjonsbar

Vi bestemte oss tidlig for at navigasjonsbaren skulle være enkel og oversiktlig med kun 3 knapper. Dette gjorde at navigasjonen ble brukervennlig og passer på at brukerne ikke blir overbelastet med informasjon. Vi valgte å kun bruke gjenkjennbare ikoner uten tekst, fordi vi selv mente at teksten enten ville ha vært for liten for svaksynte eller ville gjør ikonene mindre ved at teksten tok for mye plass.

4.4 Brukskvalitet og pålitelighet

I denne seksjonen vil vi gi en oversikt over funksjonalitet, brukervennlighet og pålitelighet til applikasjonen, samt beskrive evalueringen av disse kvalitetsegenskapene.

4.4.1 Funksjonalitet

Det viktigste formålet med applikasjonen er å gi brukeren værmeldingen til deres ønskede lokasjon. Applikasjonen gir nå bare temperatur og hvordan været er i gitt lokasjon. Den viser ikke vindstyrke eller nedbør. Dette kan anses som en svakhet i applikasjonen vår. I en senere versjon kunne man inkludert hvor sterk vinden er og hvor mye nedbør det er om det regner.

Applikasjonen har mange av de viktigste funksjonalitetene som værmelding, søk, text-to-speech og favoritter som bruker data fra Meteorologisk institutt. Applikasjonen gir derfor et grunnlag som kan benyttes for å videreutvikle og gjøre den mer funksjonelt tilpasset i henhold til tiltenkt formål.

4.4.2 Brukskvalitet

Brukskvalitet er svært viktig for vår applikasjon og er derfor forsøkt å være utformet så brukervennlig som mulig. Dette er spesielt viktig siden vår målgruppe er svaksynte. Som skrevet i designdelen i rapporten (4.3), har vi prøvd å være brukervennlig ved å bruke gjenkjennbare ikoner og et enkelt, kompakt oppsett samt stor tekst med forhåpninger om at brukeren vil kunne forstå funksjonaliteten og kunne navigere seg i applikasjonen uten noen store problemer. Dette ble testet gjennom en brukertest med brukere med forskjellige forutsetninger og alder.

I applikasjonen er det mulighet til å navigere seg til søk, vær og innstillinger. Dette mente vi var tilstrekkelig for applikasjonen vår grunnet til at vår målgruppe er svaksynte,der tankegangen

var at det var bedre å ikke overbelaste bruker med informasjon og forskjellige funksjonaliteter.

En annen funksjonalitet for å øke brukervennligheten er når applikasjonen laster inn data, så vil det i stedet for at applikasjonen bare står og henger, vil det heller vise en skjerm for innlasting. Dette gir beskjed til brukeren om at innholdet hentes, noe som gir bekreftelse at applikasjonen fungerer og at informasjon vil komme.

Til slutt, kjørte vi det innebygde programmet Lint i Android Studio, som analyserer programkoden og gir blant annet tilbakemeldinger på brukervennlighet fra brukergrensesnittet. Etter analysen så det ut til at programmet ikke fant noen problemer med brukervennligheten i brukergrensesnittet. Likevel er dette kun et verktøy, og vi har gjort manuell testing samt relevant brukertesting for å oppnå god brukervennlighet.

4.4.3 Pålitelighet

Commented [1]: Vet ikke hvor det skal, men om tiden på telefon er feil kræsjer den alltid

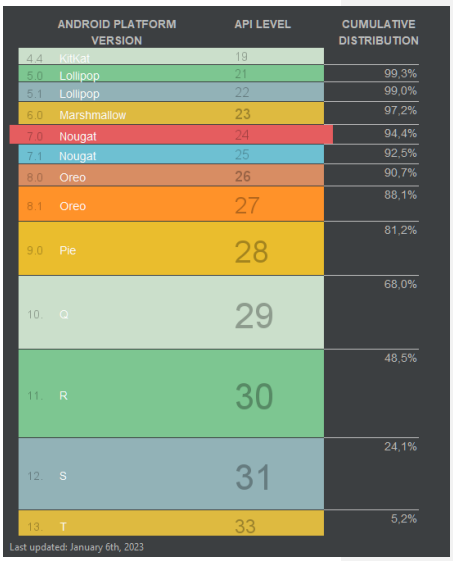
Commented [2]: Om man starter appen for fort etter oppstart eller noen ganger ved første oppstart binder ikke tts engine seg. Dermed funker tts ikke og man får en feilmelding. Appen funker som vanlig bare uten tts

Applikasjonen vår ved vanlig bruk, med internett og vanlige data fra våre datakilder, kjører feilfritt. Derfor mener vi at vår applikasjon er pålitelig. Dette undersøkte vi ved akseptansetesting av teammedlemmene. Vi hadde blant annet testet og sørget for at brukerne får riktig informasjon om været og at applikasjonen ikke kræsjer ved hyppig klikking mellom ulike skjermer. Vi la merke til et problem som vi tror kun oppstår ved kjøring på emulator. Siden MainScreen er avhengig av tid og dato fra enheten, kan utdatert dato eller feil tid på enheten føre til at applikasjonen kræsjer.

Grunnet lite tid igjen så fikset vi dessverre aldri hvordan applikasjonen oppfører seg når applikasjonen ikke klarer å få informasjon fra Location Forecast-APIet. Hittil har vi kun en catch exception som da vil vise en hvit skjerm med en error melding.

Siden Location forecast-APIet ikke kan vite hva slags vær det er lengre fram i tid, men bare temperatur, så har vi gjort det slik at det viser "?", istedenfor et blankt felt, slik at brukerne kan få vite temperaturen den dagen, selv uten noe værinformasjon.

4.5 API-nivå



Vi endte opp med å bruke API-nivå 24. Ved bruk av dette API-nivået vil de fleste Android brukere få tilgang til applikasjonen, det vil si rundt 94.4% av alle Android-enheter ifølge Android Studio. Vi hadde egentlig planlagt å bruke API-nivå 23, men søkefunksjonene ville ikke fungere og var grunnen til at vi endte på API-nivå 24.Vi hadde heller ingen behov for et høyere API-nivå for applikasjonen vår fordi applikasjonen ikke var avhengig av noen nye egenskaper eller funksjoner som krevde et høyere API-nivå. I tillegg ville et høyere API-nivå ledet til at færre Android-enheter ville kunne brukt applikasjonen vår. I tillegg fant vi ut at Google Play lagde et nytt reglement hvor nye apper må målrettes mot API-nivå 33. Derfor forsøkte vi ikke å fikse problemet med at applikasjonen ikke fungerte som det skulle på det originale API-nivået 23 ("Meet Google Play's target API level requirement")

4.6 Universell utforming

Universell utforming handler om å lage produkter og tjenester med tanke på at flest mulig brukere skal kunne bruke tjenestene på en likeverdig måte. Dette var et nøkkelprinsipp som

vi fulgte og stilte krav til underveis med tanke på målgruppen vi hadde valgt. For å sikre oss at vi jobbet mot universell utforming, brukte vi "Web Content Accessibility Guidelines" bedre kjent som WCAG, der vi tok utgangspunkt i versjon 2.1.

WCAG er et sett med punkter og krav med formål om å gjøre nettbaserte tjenester mer tilgjengelige for alle brukere. Vi sørget for at applikasjonen skulle ha et så minimalistisk utseende som mulig og at kun nødvendig informasjon ble vist uten noen overflødige animasjoner eller fargeendringer. Punktene vi fulgte var det vi i teamet tenke var de mest relevante med tanke på en applikasjon rettet mot svaksynte. Siden WCAG var laget med tanke på nettbaserte sider og ikke nødvendigvis applikasjoner på telefonenheter, så var ikke alle punktene like relevante. Noen punkter var heller ikke relevante ettersom applikasjonen ikke hadde noe innhold som krevde det. For eksempel så vil punkt 1.2.1 som krever teksting for video med lyd, ikke være relevant fordi applikasjonen vår ikke har noen form for video.

WCAG kravene kan bli delt inn i 4 hovedprinsipper med deres respektive underpunkter:

1. mulig å oppfatte
2. mulig å betjene
3. forståelig
4. robust

Det første punktet handler om at informasjonen presenteres på en slik måte at alle brukere kan oppfatte. Altså at det ikke skal vises informasjon som bare kan tolkes av en sans.

- 1.1.1 Applikasjonen tilbyr informasjon både visuelt i form av tekst og bilder, samt muntlig via text-to-speech. Applikasjonen viser ellers minimalt innhold. Det er ikke noen bilder og separate lydfiler som krever sin egen tekstboks.
- punktene under 1.3 dekker krav til kodemessig oppbygging av applikasjonen. Applikasjonen skal ha klare overskrifter, skal lett vise hvor brukeren interagerer med applikasjonen og applikasjonens hovednavigasjon skal ikke endres for å unngå forvirring hos brukeren. Applikasjonens innhold skal vises i en meningsfull rekkefølge. Dette løste vi ved at vår app har en stor og klar fontstørrelse som viser de ulike titlene på hver sine skjermer. komponenter skal være klart adskilt fra

hverandre, søkefeltet har en klar tekstlig markering om hva den gjør. Navigasjonsbaren på applikasjonen endrer seg ikke fra skjerm til skjerm- Både ikonene på knappene og størrelsen på knappene forblir det samme

- 1.4.1 Applikasjonen bruker ikke farge som eneste middel for å formidle informasjon. Applikasjonen bruker farger hovedsakelig for kontraster mellom objekter. Applikasjonen viser ikke noe fargebasert innhold som kan være tvetydig, noe som er svært for fargeblinde og svaksynte.
- 1.4.3 Applikasjonen dekker kontrastforholdet på minimum 4,5:1. Applikasjonen bruker en fargekombinasjon med lilla, svart og hvit, og vi passer på at lilla aldri møter svart siden dette ville ha vært under grensen for kontrastforholdet. Lilla på hvit gir en kontrast på 7,12:1.

Det andre punktet handler om at brukeren skal kunne betjene brukergrensesnitt, komponenter og navigering funksjoner.

- 2.2 og 2.3 applikasjonen har ingen informasjon som går bort etter en viss tid og har ingen blinkende animasjoner. Ettersom begge disse hadde vært plagsomme og gjort informasjonen mindre tydelig for en svaksynt bruker å håndtere.
- 2.4.1 Applikasjonen starter rett på værvarslingen for brukerens lokasjon, etter samtykket til bruker. Brukeren kan ellers alltid hoppe rett til hovedinnhold med home-knappen i navigasjonsbaren.
- 2.4.6 Overskriftene og undertitlene i applikasjonen beskriver formål. Søkefeltet har titel, så brukeren får en forståelse for hva den gjør. Innstillingene har titler som forklarer hva hver seksjon har og hvilke formål de har.

Det tredje punktet handler om at informasjonen som vises skal være forståelig og på en forutsigbar måte.

- 3.2.3 Navigeringsmekanismer som gjentas på flere websider innenfor et sett av websider, opptrer i samme relative rekkefølge hver gang de gjentas, med mindre brukeren selv foretar en endring. Applikasjonen vår har en navigasjonsbar som forblir lik i samme posisjon på tvers av de forskjellige skjermene.

- 3.2.4 Komponenter som har samme funksjonalitet innenfor en samling av nettsider, identifiseres på en konsekvent måte. Applikasjonen har bare 3 skjermer. Alle skjermene har unikt formede komponenter som kun oppstår på hver sin side. weatherCards vises kun på MainScreen, favorittlokasjoner vises kun på SearchScreen. og dropdown meny vises kun på SettingScreen.
- 3.3.1 Google APIene som ble brukt til applikasjonen viser kun mulige steder som forslag, på den måten kan ikke brukeren selv velge noe som ikke finnes. Dersom en feil oppstår og api-kallet mislykkes, så vil brukeren få en feilmelding som forklarer at api-kallet mislyktes.
- 3.3.3 Dersom feil blir oppdaget automatisk, får brukeren et forslag om hvordan feilen kan fikses. Applikasjonen gir forslag om å lukke og åpne applikasjonen på nytt dersom et api-kall mislykkes.
- 3.3.4 Applikasjonen samler ikke inn sensitiv informasjon fra brukeren. Den spør kun om tillatelse til den tilnærmede lokasjonen til brukeren for å vise været på gitt lokasjon.

Det fjerde punktet handler om robusthet, og er hovedsakelig sentrert rundt programkoden. Punkt 4 handler om at det skal være enkelt å bygge på og videreutvikle koden. I tillegg skal koden tolkes på en pålitelig måte. Punktene fra denne delen var ikke like relevante i vårt tilfelle siden vi utviklet en applikasjon, men selvsagt så inngår deler av punktene inn i generelt gode programmeringsvaner og konvensjoner. Alle komponentene til applikasjonen har bestemt navn og rolle, samt at det ikke er duplikater av variabelnavn og funksjoner.

4.7 APIer brukt

LocationForcast

LocationForcast er et API fra Meteorologisk institutt. APIet holder informasjon på været time for time de neste 60 timene i framtiden. Grunnen til at vi brukte dette var fordi vi trengte et API for værdata og mente at å ha værdata for 60 timer frem i tid ville vært nyttig.

Google Api (geocoding & places)

Google Place API og Google Geocoding API er kraftige verktøy som gir utviklere tilgang til stedsbasert data og tjenester. Google Place API brukes for å hente lokasjoner brukerne våre vil søke på. Vi kaller APIet hver gang bruker skriver inn en bokstav, og får dermed foreløpige forslag til steder. Google Geocoding API konverterer adresser (som gateadresser, landemerker eller postnumre) til geografiske koordinater (breddegrad og lengdegrad) og omvendt. Vi brukte det til sistnevnte frem til vi oppdaget at kostnadene ble for høye.

5. Enhetstesting

Først har vi en test for funksjonen getFavoriteLocations. Denne testen simulerer henting av favorittlokasjoner fra sharedpreferences. Den bruker Mockk for å mocke sharedpreferences og context-objektene, samt forbereder oppførselen til disse avhengighetene. Testen sjekker deretter om de hentede favorittlokasjonene stemmer overens med de forventede verdiene. Denne testen er viktig fordi den sikrer at applikasjonen vår henter favorittlokasjoner korrekt fra lagringen. Den hjelper oss med å identifisere eventuelle problemer med datahenting eller lagring.

Testen for funksjonen isFavorite følger et lignende mønster. Den tester om en lokasjon er merket som favoritt. Denne funksjonen er avgjørende ettersom den påvirker brukergrensesnittet og oppførselen til applikasjonen. Brukerne skal kunne se om deres valgte lokasjoner er merket som favoritter på en nøyaktig måte.

Deretter har vi to tester relatert til TextToSpeechViewModel. Testene testOnTextFieldValueChange og testInitialUiState. Disse testene er viktige for å sikre at vår ViewModel korrekt gjenspeiler endringer i brukergrensesnittet og starter med riktig initialtilstand. Disse testene bekrefter at vår ViewModel reagerer korrekt på brukerinteraksjoner og initialiserer med riktig tilstand, noe som sikrer en jevn brukeropplevelse.

Videre undersøker vi to tester relatert til WeatherViewModel, nemlig testInitialWeatherUiState og setDataSource changes _weatherUiState to Loading. Disse testene sikrer at ViewModelen starter med riktig initialtilstand og endrer tilstand korrekt når

datakilden oppdateres. Nøyaktige tilstandsforandringer er avgjørende for korrekt visning av data og lasting av tilstander til brukeren.

Testene fetchCoordinates og fetchLocationData omhandler nettverksoperasjoner og datahenting. Testene bruker en simulert server for å simulere nettverksrespons og sjekker om funksjonene våre håndterer responsene korrekt. Denne testingen er avgjørende for å sikre at applikasjonen vår kan håndtere forskjellige serverresponser og korrekt tolke og bruke de mottatte dataene.

Testen testGetDayOfWeek sikrer at applikasjonen vår riktig bestemmer ukedagen fra en dato. Denne funksjonen er viktig for korrekt visning av datoer til brukeren.

Til slutt er textUiStateTest og currentLocationDataTest dataklasse-tester. De sjekker at dataklassene lagrer og returnerer dataene sine korrekt. Denne testingen sikrer at dataklassene våre fungerer korrekt, og at applikasjonen vår bruker og viser disse dataene korrekt.

Oppsummert spiller hver av disse testene en viktig rolle i å sikre at forskjellige deler av applikasjonen vår fungerer som de skal. De hjelper oss med å oppdage og fikse feil tidlig i utviklingsprosessen, noe som sikrer en jevn og behagelig brukeropplevelse. Testing er en avgjørende del av Android-utvikling. Grundige, godt skrevne tester er grunnlaget for pålitelige, robuste applikasjoner.

6. Prosessdokumentasjon

6.1 Smidig utviklingsmetodikk

Den valgte smidige utviklingsmetodikken spilte en avgjørende rolle i prosjektet vårt. Utviklingsmetodikken hjalp med å få fart på startfasen av prosjektet og videre sette opp arbeidet på prosjektet på en enkel, oversiktlig og smidig måte.

Vi benyttet oss av Scrumban metodikken som er en blanding av scrum og kanban. Scrum er en metode som legger vekt på samarbeid og kontinuerlig forbedring. Scrum hjelper teams

med å administrere prosjekter og dele opp prosjektet i mindre biter som kalles sprinter. Kanban legger vekt på å visualisere arbeidet og tydelig representere arbeidet som må gjøres, samt hva status er for hver arbeidsoppgave. Scrumban er en hybrid av scrum og kanban og kombinerer prinsippene til begge. Den kombinerer strukturen og rammeverket til Scrum med visualiseringen og fleksibiliteten til Kanban. Dette var viktig for oss i teamet og betydde at vi hadde fleksibiliteten når det gjaldt sprintlengde og ansvar for hvert teammedlem. Scrumban bruker også tavler fra kanban som ga teamet en klar visuell representasjon av arbeidet som må gjøres og statusen til hver av oppgavene, slik at teammedlemmer kunne prioritere arbeidet sitt.

Hver uke hadde vi to stand-up møter og et retrospektivt møte på slutten av uken. På mandager og torsdager så tok vi opp hva vi hadde gjort, definerte nye oppgaver og delte på arbeidet for sprinten. Hvis arbeidet ikke ble ferdig til sprintslutt, ble den tatt med over til den neste iterasjonen. Vi jobbet for det meste parallelt der teammedlemmene hadde ansvar overfor egne oppgaver. Det ble også satt opp slik at flere kunne jobbe på en oppgave hvis det trengtes.

6.2 Verktøy

Under prosjektet så ble mange forskjellige verktøy tatt i bruk for planlegging og gjennomføring av prosjektet.

Discord

Som vårt viktigste kommunikasjonsverktøy så brukte vi Discord. Vi opprettet en server i Diskord med ulike kanaler for å snakke om forskjellige ting, for eksempel UI, kode, rapport osv. Vi følte kommunikasjonen gjennom Diskord funket bra. Vi brukte også Discord til digitale stand-up-møter, når vi ikke hadde tid til å møte fysisk.

Facebook

Facebook Messenger var vårt andre kommunikasjonsverktøy. Vi brukte for det meste facebook messenger til å planlegge møter og mindre viktig chat, der Discord ble brukt for det meste til diskusjon rundt prosjektet.

GitHub

GitHub brukte vi for kodeskikk og bevaring av koden samt versjonshåndtering. Hovedgrunnen til at vi brukte Github er Git. Dette ga oss muligheten til å kunne spore endringer, gå tilbake til tidligere versjoner og samarbeide sømløst med hverandre. Vi lagde forskjellige branches for å dele inn i oppgaver(lage UI og funksjoner), slik at vi senere enkelt kunne merge sammen branches til main branch når vi var ferdig.

Google drive

Google Drive var et viktig verktøy siden det var der alt annet en kode ble lagret. Google Drive ble brukt til å lagre alle dokumenter fra rapporten, Oblig 3 og notater/logg. Vi mente det var viktig å ha et sted der vi kan samle dokumentene våres

Trello

Trello ble brukt for å få en oversikt over status på ulike oppgaver vi hadde. Her så dokumenterte vi ulike oppgaver som skulle gjøres, oppgaver som pågikk og oppgaver som var ferdig. Takket være dette kunne alle ha en viss oversikt over hvor langt vi hadde kommet i prosjektet og i de forskjellige iterasjonene. Nå var det lettere å se hvem som gjorde hva for å unngå å arbeide med den samme arbeidsoppgaven.

Figma

Figma ble brukt til å lage en prototype av applikasjonen. Vi opprettet flere ulike prototyper med ulike farger og oppsett. Figma hjalp med å bestemme hvordan designet skulle se ut når det endelige produktet var ferdig.

Teams

Teams brukte vi for å få tak i veilederne våres Eirik og Endre, samt få med oss viktige informasjon som ble lagt ut av dem.

6.3 Faser

Vi har delt utviklingsprosessen vår i fem faser basert på de ulike sprintene våre og oblig 3

6.3.1 Oppstartsfasen

Valg av case og krav

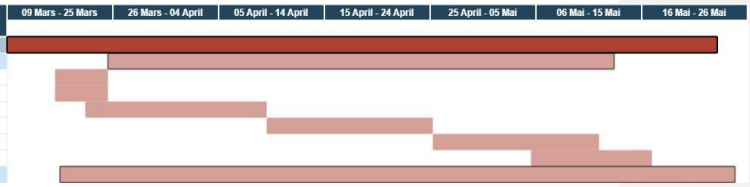
Vi sto som et team mellom case 2. strømpriskalkulator og case 3. værvarsel for svaksynte. Etter litt drøfting og et valg endte vi til slutt med case 3, værvarsel for svaksynte. Det var et par viktige grunner til dette. Vi trodde at vi kunne dra nytte av domeneekspertene innen værdata som var tilgjengelige gjennom Meteorologisk institutt. Siden værtjenesten Yr samarbeider med Meteorologisk institutt, gjorde det også mulig å få deres synspunkter på universell utforming.

Etter det satte vi opp krav og organiserte dem etter relevans. Dette dannet da grunnlaget for applikasjonen som vi senere utviklet.

Prosjektplan

Et prosjektplan ble laget for å veilede oss gjennom utviklingsprosessen. Den var åpen for at endringer kunne oppstå underveis, men det var fint å ha en helhetlig oversikt og noe å jobbe etter.

	Task Name	Duration	Start	ETA
1	Complete project execution		09.03.23	26.05.23
2	Programmering		25.03.23	10.05.23
3	Valg av Arkitektur		17.03.23	25.03.23
4	Valg av Design pattern		17.03.23	25.03.23
5	Bli ferdig med MVP		17.03.23	10.04.23
6	Prototype		10.04.23	25.04.23
7	Design		25.04.23	10.05.23
8	Bruker undersøkelse		01.05.23	20.05.23
9	Skriving av Rapport		17.03.23	25.05.23



6.3.2 Første Sprint

Sette opp GitHub

Et av grunnene til at vår første sprint i prosjektetarbeidet varte i litt over 2 uker er fordi vi hadde problemer med å sette opp Git og Github. Det å lage et github repertoar samt få koblet det opp mot git i android studio var en god del vanskeligere enn det vi så for oss. Vi møtte store problemer med noen .xml-filer som gjorde at vi ikke fikk til å pulle fra githubben, i tillegg til at det å lære seg å bruke git for oss var forvirrende og i starten utrolig tidkrevende.

Det som hjalp oss med å løse problemene våre var at vi valgte å sitte igjen sammen etter at de initielle møtene var over, for å finne ut av dette sammen. Vi lærte og feilet sammen helt til alle på gruppen skjønte hvordan man skulle beherske Git og til det funket for hvert enkelt gruppemedlem. Vi løste problemet med .xml-filene ved å putte dem i gitignore. Når omsider Git var satt opp, lært og var velfungerende var det utrolig mye enklere for oss som gruppe å gå videre i prosjektet siden vi alle kunne arbeide med det vi trengte og ville arbeide med selv. Git gjorde det mulig for alle å arbeide selvstendig eller sammen om noen ønsket det og vi er glade for at vi valgte å bruke litt ekstra lang tid på å lære oss å anvende dette systemet.

Starten av MVP

En annen grunn til at Sprint 1 ble så lang er at vi bestemte oss for å ha som et mål for sprinten var å ha en fungerende MVP så fort som mulig. Vi undervurderte hvor mye det skulle til for en fungerende MVP og innså at vi burde ha delt det opp i mindre biter på starten enn å ha alt i en iterasjon. Denne realiseringen ledet til at vi senere bestemte oss for at en ferdig og fungerende MVP kom til å vare over flere iterasjoner.

På et gruppemøte satte vi oss alle ned og diskuterte hvordan vi ønsket at applikasjonen vår skulle se ut i henhold til kravene vi hadde satt for målgruppen vår. Tidlig ble vi enige om at fokuset skulle være på funksjonalitet i henhold til svaksynte og blinde. Altså at det viktigste ikke var hvor fint alt skulle se ut, men at det skulle være så enkelt som mulig for en svaksynt bruker å bruke applikasjonen vår.

Design prototyping

Etter å ha bestemt hva MVP skulle inneholde, startet vi å lage et design som var en prototype av MainScreen for å skape en visuell fremstilling av applikasjonen ved bruk av figma. Prototypen gjorde det lettere å diskutere utformingen av applikasjonen, blant annet også å se alternative løsninger kjapt. Vi brukte prototypen som en mal for hvordan oppsettet skulle se ut i applikasjonen. Under programmeringen var det en stor fordel at de

fleste avgjørelsene angående applikasjonens utseende allerede var satt, noe som ga store fordeler.

Første iterasjon av skjermene

Vi startet med design av hovedskjermen, kalt MainScreen, som skulle vise været fordi vi

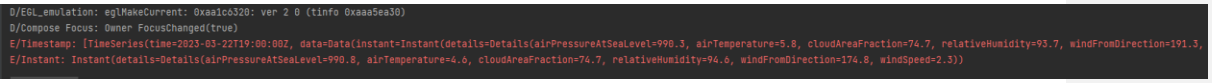
hadde en designprototype av hovedskjermen. Vi ville sikre at UI skulle fungere på ulike enheter med forskjellige skjermstørrelser. Til dette brukte vi LocalConfiguration.current til å hente bredde og høyde spesifikasjonene, som het screenHeightDp.dp og screenWidthDp.dp. Vi brukte disse variablene til å spesifisere bredden og høyden til ulike composables i applikasjonen. Dette sikret at selv på andre telefoner så ville UI layout være relativt lik. før kodingen ble det laget en grov skisse av layout (*se vedlegg 2*). Dette ble da implementert i første iterasjon av MainScreen. De andre skjermene hadde ikke dette, ettersom dette tok litt tid og teamet ønsket å ha andre skjermer for testing av navigasjon.

Alle skjermene ble planlagt for å ha relativt komponent størrelse som MainScreen, dette kommer det mer om i fjerde sprint.

Hente data fra LocationForecast

Samtidig som noen jobbet på hovedskjermen, så jobbet to andre på APIet. For å hente data fra LocationForecast så måtte vi først lage en dataklasse som skulle holde på dataen. Vi brukte et oversettingsverktøy som oversatt JSON til en kotlin dataklasse. Dette effektiviserte prosessen med å få tilgang til og manipulere dataene, noe som gjorde det mer håndterbart og effektivt for teamet vårt. Vi sendte en HTTP-forespørsel til APIet og mottok et JSON-response som inneholdt værinformasjon for de neste timene og dagene. For å sjekke om de mottatte dataene og sikre at de er korrekte, brukte vi LogCat, et kraftig feilsøkingsverktøy levert av Android Studio. Ved å logge JSON-svaret i LogCat-konsollen, kunne vi inspisere strukturen og innholdet i dataene. Vi så for oss å også hente MetAlerts

Commented [3]: Velg om vi skal ha det med



og TextForecast. TextForecast så vi at kom til å bli overflødig å ha med, da vi fikk nok informasjon av LocationForecast til å lage egne setninger, som igjen kunne bli brukt til text-to-speech. MetAlerts var et ønske, men vi tenkte at dette var bedre å nedprioritere til vi hadde en mer fungerende app.

6.3.3 Andre Sprint

Andre sprint var betydelig kortere og mindre enn sprint 1. Vi planla at denne sprinten skulle være med på å fikse opp ting vi slet med i den første sprinten og bygge videre på det vi allerede var ferdige med. Denne sprinten var ekstra liten siden den varte over påsken der samtlige deler av teamet var bortreist og tok seg ferie.

Data på skjermen

I starten av andre sprint så fikk vi til å få LocationForcast informasjonen på skjermen. Vi fikk til å vise lokasjonen, datoen og temperaturen. På dette punktet var alt fremstilt enkelt og midlertidig med bare det grunnleggende rammeverket. Vi innså at om vi skulle vite hva været er på en lokasjon så måtte vi vite hva koordinatene til lokasjonen var og innså at fåtallet av brukere ikke vet koordinatene til de stedene de ønsket å se værdata, så vi prøvde å finne en løsning på dette.

Søkefunksjon første iterasjon

Den første iterasjonen av søkefunksjonaliteten var først planlagt til å fullføres med Kartverkets åpne API for søk etter stedsnavn (*se vedlegg 11*). Det er viktig å nevne at Yr bruker Kartverkets data for stedsnavn i Norge. Vi manglet en funksjon som ga bruker forslag da det ble søkt etter stedsnavn. Vår første idé var å bruke Kartverket sitt API. Vi fikk til å kalle på APIet for å få tilbake data, og forsøkte å lage en søkefunksjon som tok inn input via et TextField og fortløpende ga forslag til stedsnavn (autoutfylling) for hver bokstav skrevet inn. Dette viste seg å være en ganske vanskelig oppgave. I tillegg tenkte vi at det å gjøre så mange API-kall under hvert søk var lite effektivt og førte til unødvendig mye datatrafikk og prosesseringstid for applikasjonen under kjøring. APIet hadde en begrensning på maks 500 treff per side/request, så det var heller ikke mulig å gjøre et stort API-kall under første kjøring av appen for å hente inn alle stedsnavn fra Kartverket. Vi begynte derfor å se etter en annen løsning.

Vårt neste forsøk på å implementere søkefunksjonalitet startet med en idé om å ha alle stedsnavn lagret inne i appen for å begrense antallet API-kall under kjøring. Vi lagde et script i python vi kjørte utenfor appen, som gjorde et stort antall API-kall (med maks 500 treff per side), forsøkte å hente ut absolutt alle stedsnavn i Norge samt deres lengde- og breddegrader fra Kartverket. Deretter skrev python-scriptet alle stedsnavn og tilhørende koordinater inn i et HashMap til en .kt-fil vi kunne legge til i prosjektet. På denne måten hadde vi alle stedsnavnene lagret når appen startet opp, og vi kunne implementere en

søkealgoritme som hentet data fra denne kotlin-filen fortløpende mens brukeren skrev inn søk. Hovedproblemet her var at det kom noen uforutsette problemer med filen som vi ikke helt skjønte, men antok det var fordi filen ble utrolig lang (på grunn av antallet stedsnavn) og vanskelig å jobbe med. I tillegg ble vi usikre på om python-scriptet i det hele tatt hadde klart å få med alle stedsnavn, og vi ønsket å bruke tid på andre ting enn å lage en veldig effektiv søkealgoritme. Det var grunnene til at vi omsider gikk vekk fra denne ideen og lette etter andre alternativer.

6.3.4 Tredje Sprint

Loading Screen

I tredje sprint ble det implementert et enkelt design for når WeatherUiState-objektet i MainScreen har status "Loading". Designet består av en enkel Text-Composable med teksten "Loading", over en CircularProgressIndicator-Composable som viser animasjon for innlasting når appen laster. Teamet bestemte også at de ville implementere et bakgrunnsbilde for LoadingScreen. Vi brukte da gimp til å lage flere varianter av bakgrunnsbilde(*se vedlegg 4*). Bildene ble presentert til gruppen og ett av bildene ble valgt. Det var derimot en del problemer med å implementere bakgrunnsbilder, og gruppen bestemte til slutt for å forkaste ideen. Utover dette ble designet til innlastings-animasjonen aldri endret, fordi det ble nedprioritert ganske tidlig i utviklingsprosessen.

Google API

Etter mye testing med filene og få SearchScreen til å funke, fant vi til slutt ut av at Google sin API tjeneste ble det rette å bruke for oss. Ved å implementere Places Api kunne vi nå hente koordinater fra navn, hente stedsnavn ved å sende inn koordinater og hente forslag fortløpende. Den siste funksjonen er hovedgrunnen til at vi byttet til Google sine løsninger. Et problem vi naturligvis støtte på er hva slags forslag vi får, der man får opp generelle steder før man har skrevet omtrent 3 bokstaver. Så det vi ønsket var å finne en måte å prioritere forslag ut fra hvilke land bruker oppholder seg i. Dette lykkes vi dessverre ikke med.

TextToSpeech

Etter store kostnader på google APIene våre ville vi gå en annen vei med TTS. Ved å implementere Android sin egen TextToSpeech objekt ga oss denne muligheten. Vi støtte på problemer da vi ikke hadde en TTS-engine installert på emulatorene våre. Når vi fikk dette på plass kunne vi ta i bruk TextToSpeech Viewmodel. Vi så hensikten med å starte denne viewmodellen i mainactivity og deretter sende den videre til skjermene. Slik ble det startet tidlig nok slik at TTS hadde blitt aktivert når loading screens var over.

Viewmodels

Siden de forskjellige skjermene våre trenger samme ViewModel og aktivt kaller på metoder fra den, valgte vi å opprette ViewModels i MainActivity. Ved å sende dem som parametere kunne vi da heller bruke collectAsState der vi trengte dem. Spesielt med TTS-View Modelen fikk vi store komplikasjoner med å opprette denne i hver skjerm.

MVP ferdig

Vi ble ferdig med en fungerende MVP og endte opp med 3 skjermer i applikasjonen. Første skjerm var SearchScreen, som er søkeskjermen. Denne ble designet med overskriften "Lokasjon" på toppen, en søkebar der brukere skulle kunne søke etter en lokasjon de ønsket å se været til, en søkeknapp og en navigasjonsbar. Søkebaren skulle oppfylle WCAG 1.3.1, og derfor la vi til en beskrivende tekst i søkebaren, som var stor nok til å kunne enkelt leses. Søkeknappen endte vi opp med å forkaste. Grunnen til det var at vi fant den overflødig, da vi endte opp med å kunne trykke på lokasjon rett fra forslagslisten. Samt la forslagslisten seg over søkeknappen, noe som gjorde skjermen uoversiktlig.

MainScreen er skjermen som skulle vise hovedmålet vårt med applikasjonen, altså presentere værdataen for gitt lokasjon. I følge våre MVP- krav, skulle skjermen i første omgang vise værdata for gitt lokasjon for hver time. Dette gjorde vi ved å legge til Cards-komponenten og legge disse til i en LazyRow. Videre tilpasset vi fargevalgene i applikasjonen, slik at applikasjonen kunne oppfylle WCAG 2.1 nivå AA, som omhandler kontraster.

SettingScreen skulle være en skjerm som viser ulik informasjon, og funksjoner for å bytte fargekontraster for hele applikasjonen, med hensyn til fargeblindhet. I skjermen finnes det Cards som utvides med mer informasjon ved et trykk. Skjermen skulle ha informasjon om applikasjonen, ikoner vi benyttet oss av, ulike API vi tok nytte av og funksjonene for fargekontraster. Funksjonene for fargekontrastene ble ikke implementert.

6.3.5 Fjerde Sprint

Dynamisk UI-layout

Vi innså at skjermene og knappene var ulike størrelser basert på hvilken enhet appen ble emulert på. I første sprint ble MainScreen utviklet med en relativ størrelse, slik at den kunne tilpasses forskjellige skjermstørrelser. Imidlertid ble ikke de to andre skjermene utviklet med samme tilnærming. Planen var å ha en god utforming for alle skjermstørrelsene, og dette ble diskutert tidlig i prosjektet. Akkurat som MainScreen, ble andre skjermer også skissert med prosenter før koding (*se vedlegg 5*). Dessverre oppstod det problemer med Android Studio (*se vedlegg 6*), som hindret gruppemedlemmet, som var ansvarlig for layouten, fra å kjøre applikasjonen og fikse formateringsproblemet. Disse problemene varte i to uker for dette gruppemedlemmet, noe som forsinket arbeidet med å fikse layouten.

Ellers på grunn av prioriteringer i prosjektet ble det besluttet at resten av gruppen skulle fokusere på å implementere andre funksjonaliteter som ble ansett som viktigere da. Det var fortsatt et klart behov for å sikre applikasjonens robusthet og sikre at informasjonen ikke gikk tapt på forskjellige skjermstørrelser, men det ble ansett som mindre kritisk på det tidspunktet.

Vi slet litt med å finne en løsning for tekst som skal tilpasses etter størrelsen på sin parent-container. Løsningen ble å lage en funksjon AutoResizedText som tok inn en string som argument, og endret skriftstørrelsen på denne helt til den fylte parent-containeren horisontalt. Denne løsningen ble ikke perfekt, og noen Text-Composables overflower fortsatt vertikalt. Med mer tid hadde vi fokusert på å forbedre denne funksjonen, eller funnet en bedre løsning for å skalere tekst.

Alle elementene på MainScreen og SearchScreen har størrelse relativ til skjermstørrelse og andre Composables. Vi rakk ikke å implementere dette for SettingsScreen bortsett fra overskriften, men siden den inneholder så få elementer ser utformingen grei ut uavhengig av skjermstørrelse.

Oppdatert UI - elementer

Videreutviklingen av vårt design innebar å legge til Cards i MainScreen, som viser værdataen for dager frem i tid. Disse kortene kan trykkes på for å se time for time den angitte dagen. Videre, i SearchScreen, implementerte vi inn Cards som representerte listen med lagrede favorittlokasjoner. Disse kortene kan brukere trykke på for å se værdataen for den bestemte favorittlokasjonen. For å kunne lagre favorittlokasjoner la vi til en lagringsknapp. Knappen er representert som en fylt stjerne ved lagring og ikke-fylt stjerne ved ingen lagring. Dette viser bruker om stedet er lagret eller ikke.

Favorittlokasjoner - første iterasjon

Neste funksjonalitet vi la til i applikasjonen var en mulighet for brukere å lagre sine favorittlokasjoner. I første omgang kunne bruker lagre en lokasjon, der stedet ble lagret i et Card på SearchScreen. Hvis bruker lagret en ny lokasjon, ville den gamle lokasjonen byttes ut med den nye. Videre skulle funksjonen skulle kunne lagre opp til 4 steder, samt å kunne fjerne favorittlokasjoner og kunne trykke på et spesifikt Card for å vise værdata til den gitte lokasjonen.

Brukertesting av applikasjonene

Det var i fjerde sprint i starten av mai da vår første brukertesting ble gjennomført. For mer informasjon gå til (5.4 Brukerinvolvering)

6.3.5 Innspurtsfasen

Finpussing av applikasjon

På dette punktet nærmet vi oss siste del av prosjektet og implementerte text-to-speech ved å lage en separat viewmodel som kalles på ved et trykk på de forskjellige kortene der informasjonen blir lest opp. Resten av sprinten var til å fikse opp i mangler på implementasjonen vi allerede hadde, samt å rydde opp i koden vår. Scalability var en prioritet siden vi fant ut at deler av informasjonen som skulle fremstilles ikke var konsistent ved bruk av forskjellige emulatorer av forskjellige størrelser. Vi testet dette med å kjøre applikasjonen i Nexus 10 emulatoren som er en større skjerm, samt Nexus One som har en mye mindre skjerm. Dette problemet løste vi ved å bruke prosentandeler for størrelser på cards og bottomBar. På denne måten vil elementene alltid være en like stor del av skjermen uavhengig av størrelsen på skjermen. Vår favoriteLocation-funksjon manglet navigasjon og fungerte kun som en visuell fremstilling av hvilke lokasjoner som var lagret som favoritter. Denne representasjonen ble gjort om til clickable cards som sendte den lagrede lokasjonen, som videre oppdaterte MainScreen og fremstilte værdata til lokasjonen.

Vi så og at programkoden vår hadde ganske mange mindre advarsler som inkluderte skrivefeil siden vi hadde skrevet kommentarer på norsk og ikke engelsk, samt at vi hadde en del ubrukte variabler. De ubrukte variablene ble fjernet og alle kommentarene til programmet ble oversatt til engelsk. Vi ønsket også at all koden vår skulle være skrevet konsist gjennom. Altså at alle variabelnavn og funksjonskall skulle endres til CamelCase.

Rapportskriving

Selv om vi hadde skrevet en god del av rapporten før fristen, var rapporten en god del av innspurtsfasen. Vi hadde logget og skrevet en god del av utviklingsprosessen og skrevet litt av presentasjonen av caset og brukerdokumentasjonen gjennom de forskjellige iterasjonene våres, men manglet fortsatt en god del. Her ble det viktig å fordele deler av rapporten for å fullføre den i et høyere tempo. Etter at gode deler av rapporten hadde blitt skrevet så leste noen av teammedlemmene rapporten for å forsørge god flyt og at rapporten ikke manglet noe.

6.4 Brukerinvolvering

For å sikre brukervennligheten i applikasjonen, anser vi brukerinvolvering som avgjørende. Vi tok i bruk brukertester, som ga verdifulle bidrag til utviklingen.

6.4.1 Brukertest

Hensikten med brukertesten var å få tilbakemeldinger fra svaksynte brukere angående design og navigasjon for å vurdere brukervennligheten. Opprinnelig planla vi å gjennomføre flere iterasjoner av brukertester for å justere applikasjonen etter behov, men på grunn av begrenset tid ble dette utfordrende.

Planlegging av brukertesting

Ettersom vi hadde svaksynte brukere som målgruppe så var det viktig å ha brukerinvolvering for å sikre at applikasjonens utvikling gikk i riktig retning. Vi fikk fort vite at å ha en veldig spesifikk målgruppe også brakte sine utfordringer. Det viste seg å være utfordrende å oppnå kontakt med svaksynte folk. Denne delen beskriver vår planlegging og hvordan brukertesten endte opp til slutt.

Opprette kontakt med organisasjoner og grupper

Den originale planen var å kontakte Blindeforbundet og spørre om det var mulig å ha et par frivillige brukertester med diverse svaksynte mennesker. Dette viste seg å være ganske utfordrende ettersom vi fikk vite at Blindeforbundet ofte blir ringt opp av studenter for diverse undersøkelser. Vi antok grunnet høy etterspørsel og sensitiv personopplysning av den spesifikke målgruppen vår, at det var grunnen til hvorfor blindeforbundet ikke var veldig begeistret over så mange forespørsler. Dette fikk vi også vite fra diverse gruppelærere og andre grupper vi snakket med som også prøvde å kontakte Blindeforbundet tidligere.

Vi fikk så vite at vi burde prøve å kontakte diverse grupper på sosiale medier, hovedsakelig Facebook. Et av gruppemedlemmene prøvde å bli med i en av gruppene for å se om det var noen frivillige, men fikk ikke tilgang av administratorene. Det tok ellers lang tid, ofte mer enn en uke, for å få svar både fra Blindeforbundet og andre grupper. Dette var ikke bra ettersom vi hadde begrenset tid og mål med å få til mer enn en undersøkelse.

Det var på dette tidspunktet gruppen bestemte seg for å se etter alternative metoder for brukerinvolvering. Blant annet familiemedlemmer, men per definisjon så hadde ingen i gruppen svaksynte familiemedlemmer de kunne kontakte. Vi kontaktet også en domeneekspert hos Met for å få innblikk i hvordan Yr håndterer universell utforming og WCAG for sin hjemmeside. Gruppen ville fortsatt ha en brukertest, ettersom dette var veldig viktig for vår applikasjon. Vi hadde ellers en plan B om alt annet ikke lot seg gjøre. En person i gruppen fikk vite tidlig i prosjektet om at det finnes spesiallagde briller som simulerer diverse synsproblemer, som vi kunne låne fra en foreleser/professor som jobbet hos IFI.

Briller for svaksynte

Et problem vi møtte var at denne informasjonen var gitt på starten av prosjektperioden av en gruppelærer. Dette betydde at ingen i gruppen husket navnet til personen. Vi måtte både sende meldinger til gruppelærere, IFI studieadministrasjonen og evt spørre resepsjonen ved å gi en beskrivelse av personens stilling og hvor hen jobbet. Etter noen dager klarte vi å finne navnet hennes som var Tone Bratteteig. Teamet sendte henne flere meldinger om å eventuelt låne brillene for våre undersøkelser. Dette tok lengre tid enn forventet, men til slutt fikk vi kontakt og besøkte en av hennes forelesninger og forklarte hva vårt mål var. Vi

fikk lånt brillene i en uke. De kom i en koffert med beskrivelse og bruksanvisning. Det var 6 briller som hver simulerte ulike synsproblemer.

Med dette begynte vi å planlegge hvordan undersøkelsen skulle utføres. Det ble konkludert at noen av brillene ikke var så relevante med tanke på applikasjonen vår og målgruppen vi hadde valgt. Noen av brillene simulerte ekstreme situasjoner av sykdommer som diabetes som ikke har blitt behandlet og ødelagt synet. Andre ting som for eksempel retinal detachment ble heller ikke tatt med, siden det ikke forekom like mye som noen av de andre og at det ofte oppstår av sykdommer og skader. Vi valgte å fokusere på briller som simulerte de mest vanlige formene for svaksynthet og som ellers var mer relatert til alder enn sykdommer. Valgene våre ble da Cataract, Glaucoma og macular degeneration. Vi følte at disse var de mest relevante av de tilgjengelige brillene for vår målgruppe. De synsvekkelsene vi endte opp med forekom oftest der spesielt cataract var vanlig og et aldersrelatert problem.

Etter at brillene ble valgt bestemte vi oss for å teste på en gruppe av 6 personer. Disse 6 personene skulle helst ha ulike yrkesbakgrunn, utdanningsbakgrunn og alder slik at vi kunne få tilbakemeldinger fra en mer divers gruppe brukere. Alle de 6 personene var bekjente fra gruppen, enten kollegaer, venner eller familiemedlemmer. Vi passet på at alle samtykket til hva slags data vi samlet inn og at ingen av de 6 test brukerne så applikasjonen før selve testingen med brillene. Dette gjorde vi slik at det skulle bli så autentisk som mulig. Vi hadde 3 briller som skulle brukes og 6 brukere. Hver brille ble testet 2 ganger av 2 ulike brukere.

Utførelse av brukertesting

2 personer fra teamet utførte undersøkelsene. Selve undersøkelsene og datainnsamlingen tok 2 dager ettersom ikke alle var ledige på samme dag. Et møtested og tidspunkt ble avtalt på forhånd og gjennomføring av brukertest ble gjort ansikt til ansikt, og en om gangen. Før testing så ble hver bruker informert om hvordan de skulle bruke brillene, hvordan undersøkelsen skulle foregå med spørsmål og svar, samt at dette ble satt pris på om brukeren tenkte høyt under undersøkelsen.

Brukeren ble spurt om å gjøre en rekke oppgaver på applikasjonen. Oppgavene var delt opp til hver sin side av applikasjonen og spurte om hva som var synlig, leselig og hva som ikke var. Hva de tenkte ulike funksjoner på applikasjonen gjorde, hvor intuitivt det var, samt hva de likte og hva de hadde likt å forbedre/endre på veien videre. Oppgavene var blant annet:

- Les lokasjonene, les tiden, fortell hva slags vær det er, les temperaturen
- Sjekk været 22:00
- Sjekk været for neste dag
- Sjekk været i Bergen
- Prøv å favorisere lokasjonen din

Resultatene ble notert både på mobil og pc i form av stikkord og nøkkelpunkt. branchen som ble brukt under undersøkelsen het Feature/buttons/fix-button-layout. (*se vedlegg 3*)

Resultat

Undersøkelsen ga mye nyttig i form av tilbakemeldinger og eventuelt hva vi måtte fikse på videre. I denne delen blir det kort gått gjennom hovedpunktene som kom opp fra flere brukere i tillegg til mer unike tilbakemeldinger.

Etter at all data ble samlet kunne vi se at det var punkter som gjentok seg hos flere brukere. De fleste kunne se lokasjonens navn, værikonet og temperaturen. Det var en bruker i 40 årene som jobbet kundeservice som slet med å se teksten til lokasjonen og syntes at det gråe sky ikonet ikke var lette å skille fra den hvite bakgrunnen. Alle utenom en bruker klarte ikke å se tiden. Brukeren som klarte det fortalte at det var utfordrende og at de måtte konsentrere seg en del for å få det til. Angående ikonene på knappene så skjønte alle

brukerne hva ikonet til SettingScreen betydde. Nesten alle skjønte hva SearchScreen ikonet betydde, utenom en av testbrukerne i 60 årene som tolket det som en forstørrelsesfunksjon for teksten. Den samme personen nevnte også at brillene de fikk for glaucoma simulerte en alvorlig og ubehandlet form for glaucoma. mange av brukerne synes ikonet for Mainscreen var tvetydig. brukerne var heller ikke klar over at man kunne swipe kortene horisontalt for å få værvarsel for de neste timene eller de neste dagene. Det var kun en bruker som fant dette ut, en yngre person i 20-årene som også studerer IT. Ingen visste om knappen for lagring av favorittlokasjoner og hva den gjorde. På SettingScreen så kunne brukerne lese titlene på kortene, men ikke selve innholdet når man åpnet kortene. Alle brukerne sa også at de hadde likt en større skrift på søkefeltet og forslagslisten. Brukerne likte ellers de sterke fargekontrastene. Spesielt kontrasten med svart tekst på hvit bakgrunn. En bruker sa også at de likte hvor simplistisk oppsettet til applikasjonen var og at det gjorde det mindre belastende.

Etter at all dataen var behandlet så kom vi frem til det følgende: Endre skrift, skriftstørrelsen til tid, implementere en text-to-speech funksjon som leser opp kortene og forklarer hva brukeren kan gjøre, endre på lagringsknappen, øke skriftstørrelsen på søkefeltet og søkeforslagene, samt endre MainScreen-ikonet til noe mindre tvetydig. Vi tok også hensyn til at brillene simulerte, i tilfellet til glaucoma, er et langvarig og ubehandlet stadie som var veldig ekstremt.

6.5 Endring av kravspesifikasjoner

Noen av kravene vi endret prioritet på eller fjernet:

Applikasjonen skal ha kontrast moduser (endret fra må ha til kanskje)

Vi innså at vi allerede hadde designet applikasjonen med god nok kontrast, slik at det å ha kontrastbytting ble mindre prioritert.

Applikasjonen skal ha fargeblind modus (endret fra å bør ha til kanskje)

På grunn av lite tid og at målgruppen allerede hadde andre behov så ble fargeblind modus nedprioritert. Planen var ellers å benytte oss av Material 3 og lage flere color pallets som man kunne bytte til via settings.

Applikasjonen skal ha skjermrotasjon(endret fra å må ha til å ikke ha)

Ut fra kodekravene tolket gruppen at vi måtte ha med rotasjon. Vi fikk senere vite av veilederne våre at dette ikke var et krav så lenge vi ikke tillot applikasjonen å rotere og valgte å prioritere andre ting med tanke på leveringsfrist.

Applikasjonen skal ha en animasjon på MainScreen som viser at weather cards kan bli bladd bortover'

Dette ble ikke implementert etter at text-to-speech ble implementert fordi tekst-tos-speech forteller at det er mulig å bla mellom kortene.

7. Refleksjon -ettertid

7.1 Jobbe som team

Hele teamet har vært fornøyde med hvordan vi har håndtert prosjektet. Alt fra hvordan vi har blitt enige om hvilke valg vi måtte ta til den generelle gruppedynamikken. Vi har hatt få til ingen uenigheter som ikke har løst seg nesten umiddelbart. Vi er svært fornøyde med måten vi har behandlet hverandre og respektert hverandres tid og meninger i henhold til prosjektet.
Arbeidsfordeling ble ganske raskt naturlig fordelt ettersom hva de forskjellige teammedlemmene ønsket å jobbe med og etter hvilke styrker og svakheter de selv hadde. Dette resulterte i at noen medlemmer ble veldig backend-orienterte, noen veldig front-end-orienterte og andre mer rapport-orienterte. Dette ledet til både fordeler og utfordringer. Hovedproblemet rundt dette var at det ble veldig utfordrende å sette seg inn i noe som man ikke hadde begynt med i utgangspunktet siden det for eksempel var vanskelig for en som stort sett hadde jobbet mye med front-end å sette seg inn i hva de som hadde jobbet backend

hadde laget. Det positive rundt dette var at de som for eksempel jobbet mer backend hadde stor kontroll over systemet siden det var i hovedsak det de hadde jobbet med gjennom prosjektperioden. De visste inn og ut systemet de hadde laget og om det kom fram problemer rundt det visste de fort hvordan det skulle fikses eller endres. I tillegg til dette ble arbeidsfordeling gjennom prosjektperioden ganske låst siden det ble veldig naturlig at backend-arbeiderne naturligvis tok de arbeidsoppgavene som var mer backend-orienterte.

7.2 Fordeler og ulemper med teamarbeidet

Vi møtte på både fordeler og ulemper gjennom prosjektperioden både i hvordan det var å jobbe som team, samt problemer med å løse oppgaven vi hadde valgt.

7.2.1 Utfordringer

I startfasen møtte vi naturligvis en del problemer rundt hvordan det var å jobbe som et team. Hvordan arbeidsfordeling skulle planlegges, samt hvordan standups og teammøter generelt skulle holdes.

- Standups var en utfordring vi hadde helt i starten. Flere medlemmer snakket over hverandre som oftest resulterte i lange møter der veldig mye unødvendig ble tatt opp. Dette ledet og til at mye av de relevante punktene vi tok opp som var viktig å få med seg, ble ofte overskygget. Dette var et problem når teamet skulle hjem og arbeide individuelt, men kanskje ikke fikk med seg all den relevante informasjonen de trengte for å fullføre oppgavene de skulle etter det som var avtalt. Et resultat av dette var at det ble en del frem og tilbake og fiksing av ting som kunne vært gjort mye enklere og raskere om vi hadde hatt bedre strukturerte møter. Dette fikset vi etterhvert ved å ha faste scrum-masters som brukte 15-20 minutter på starten av møtet til at alle fikk fortalt og fått med seg det viktigste som måtte på plass det møtet. Etter det var det tid til parprogrammering og andre samtaler. Denne endringen gjorde at alle fikk frem det de trengte, fikk med seg de trengte og var klare for å jobbe med det de trengte på den måten de skulle resten av uken.

- Trello brukte vi for å visuelt representere hvilke arbeidsoppgaver som skulle gjøres samt hvem som skulle fullføre dem. Utover prosjektperioden ble det dessverre litt mer uklart om hvem som gjorde hva grunnet uklarheter i kommunikasjon samt latskap. En konsekvens med dette var at det hendte at flere teammedlemmer jobbet hver for seg på samme oppgave, eller at noen hadde skrevet opp at de skulle fullføre noe uten å ha gjort det som gjorde at flere arbeidsoppgaver ble hengende etter til nyere sprinter. Det å velge seg en arbeidsoppgave etter et møte ble og litt vanskelig grunnet at man kunne være usikker om et teammedlem allerede hadde begynt på det eller allerede hadde fullført det, men ikke skrevet seg opp på trelloboardet.

- Brukertester var utfordrende for oss å gjennomføre grunnet at vi ikke hadde informasjon på noen kandidater som kunne undersøkes og gi oss tilbakemeldinger på arbeidet vårt. Vi sendte flere mails til blant annet yr.no og blindeforbundet, men de stod hengende i ukesvis og da vi endelig fikk svar var de svært lite samarbeidsvillige. Dette ledet til mye venting og usikkerhet om hvordan vi kunne utføre dette. Heldigvis visste en av teammedlemmene våre om en foreleser som hadde briller som representerte forskjellige typer synssvakhet av høy grad. Etter noen uker med venting fikk vi da lånt disse brillene og fikk holdt brukertester på brukere som var uavhengige av prosjekter vårt og ga fine tilbakemeldinger vi tok til oss.

- Valg av metode for lokasjon-henting og geocoding var en lang prosess. Først forsøkte vi å oss på å bruke Kartverkets API for å løse dette. Kartverkets stedsnavn benyttes av blant annet Yr.no sine tjenester.Løsningen vår var å lage et python-script som hentet alle stedsnavn og tilhørende koordinater fra Kartverket og skrev disse inn i et HashMap til en .kt-fil. Formålet med denne måten å lagre stedsnavn og koordinater på, var å minimere antall API-kall under kjøring, og det ga oss mulighet til å slå opp på stedsnavn og komme med søkeforslag veldig raskt. Problemet var at filen ble for lang og dette ledet til flere problemer.

  Dette ledet oss videre til Google sitt Places-API. Google Places API-et gjorde at vi enkelt kunne hente koordinater fra navn, hente stedsnavn ved å sende inn koordinater og hente forslag fortløpende. Dette passet perfekt til våre behov, men

kom ikke uten en pris. Hovedproblemet vi måtte akseptere med Google sine APIer var at de ikke var gratis. Vi hadde 3100 kr gratis å bruke av APIer før det kom til å begynne å koste oss penger. Etter noen uker sjekket vi og innså at vi allerede hadde brukt 1400 kr. Etter litt feilsøking kom vi til konklusjonen at det var Googles geocoding-API som kostet oss mye, ettersom places-APIet kun hadde brukt 50 kr. Vi endret dette til å kun returnere "Unknown" som placeholder istedenfor GetLocationName(). Denne endringen gjorde at vi gikk fra å bruke 570 kr den ene dagen til 6 kr den neste. Hadde vi hatt mer tid, hadde vi byttet ut dette APIet med et mer økonomisk vennlig API eller endret funksjonen til å være mer sparsom på API-kallene den utfører.

7.2.2 Fordeler

- Github var noe som funket veldig bra for oss gjennom prosjektperioden. I startfasen hadde vi naturlig en del problemer og forvirring med å lære hvordan man brukte dette verktøyet for å håndtere de forskjellige versjonene av applikasjonen vår. Et av problemene dette kom med var når et av teammedlemmene våre klarte å pulle en versjon før det ble pushet og tapte arbeidet sitt. Etter startfasen gikk dette nesten helt feilfritt grunnet til at noen av teammedlemmene satte seg ordentlig inn i hvordan verktøyet fungerte for å hjelpe når det oppstod mindre problemer, samt tok seg av merging osv. Dette gjorde merging av branches en enkel prosedyre i et steg som ofte kan føre til større problemer hos team som er nye brukere av verktøyet.

8. Referanser

- *Årets caser*. (2023, January 18). IN2000, from https://in2000.met.no/2023/
- *Meet Google Play's target API level requirement*. (n.d.). Android Developers, from https://developer.android.com/google/play/requirements/target-sdk
- *WCAG-standarden*. (n.d.). Tilsynet for universell utforming av ikt, from https://www.uutilsynet.no/wcag-standarden/wcag-standarden/86?

9. Vedlegg

Vedlegg av samtykkeskjema

vedlegg 1 (samtykkeskjema)

Oslo / 06.05.23

Vil du delta i studentprosjektet Weather Sense?

Vi er en gruppe studenter i emnet *IN2000 – Software Engineering med prosjektarbeid* ved Institutt for informatikk ved Universitetet i Oslo. Med dette skrivet ønsker jeg å informere hva prosjektet vårt har som formål, spørre deg om du vil delta i prosjektet, samt berette hva deltagelse vil innebære for deg.

Formål
Formålet med vårt prosjekt er å lage en applikasjon som er designet for svaksynte. I forbindelse med at jeg konkret ønsker å lære mer om hva du synes om appen vår. Formålet med brukertesting er å forstå dine behov og din mening om applikasjonen, slik at vårt team kan endre designet basert på ditt behov.

Deltakelse
Du blir spurt om å delta fordi du faller innenfor min målgruppe, definert som svak synte. Dersom du velger å delta ønsker jeg å benytte lydintervju av deg i min datainnsamling. Brukertesting vil vare i 15-20 min, og jeg kommer til å ta notater fra brukertesting.

Frivillig deltakelse
Det er frivillig å delta i mitt studentprosjekt. Du kan når som helst avslutte brukertesten eller trekke tilbake informasjon som er gitt. Du kan når som helst velge å trekke samtykket uten å måtte oppgi grunn. Dersom samtykket trekkes vil eventuelle personopplysninger som er innsamlet om deg slettes og det vil ikke innebære noen negative konsekvenser for deg at du velger å trekke ditt samtykke.

Personvern: innsamling, oppbevaring, behandling og bruk av dine opplysninger
Ingen sensitive personopplysninger (jf. Personvernforordningens artikkel 9 og 10) vil bli innsamlet. Personlige opplysninger om deg vil kun benyttes til formålene beskrevet i dette informasjonsskrivet. Jeg behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

Personlige opplysning innsamlet vil bli anonymisert i transkriberingen og rapporteringen senest 26.05.23; ingen andre enn jeg, ei heller min fagkontakt Endre Valen og Eirik Langholm få vite hvem som er blitt brukertestet, og det som oppbevares av anonymisert rapportering fra brukertestingen vil følge Universitetet i Oslo sine rutiner for sikker oppbevaring.

Navn og kontaktinformasjon erstattes med pseudonymer. Brukertestingen vil kun behandles og kan ettersendes deg ved ønske. Dataen som oppbevares, inkludert anonymisert data, vil ikke bli publisert og vil heller ikke kunne tilbakeføres til deg.

Hva skjer med innsamlet data når studentprosjektet avsluttes?
Alle notater av brukertestingen blir slettes senest 13.06.23. Dette gjelder også anonymiserte og avidentifiserte opplysninger om deg.

Rettigheter
Vi behandler opplysninger om deg basert på ditt samtykke. Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om deg,
- å få slettet personopplysninger om deg, og
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger.

Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med Mohamed eller min universitetsansatte fagkontakt Endre Valen eller Eirik Langholm på e-post endrelv@ifi.uio.no, eirikjl@ifi.uio.no

Før brukertestingen begynner ber jeg deg om å samtykke i deltagelsen ved å undertegne på at du har lest og forstått informasjonen på dette arket, og ønsker å stille opp til lydintervju.

Med vennlig hilsen
Mohamed Muhuyadin
95520218, mohamejm@uio.no

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om studentprosjektet Weather Sense, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i lydintervju

Jeg samtykker til at mine opplysninger behandles frem til studentprosjektet er avsluttet.

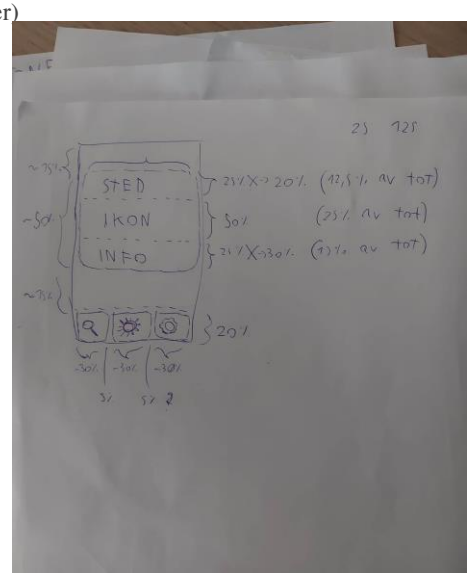
_____
Sted og dato

_____
Fullt navn

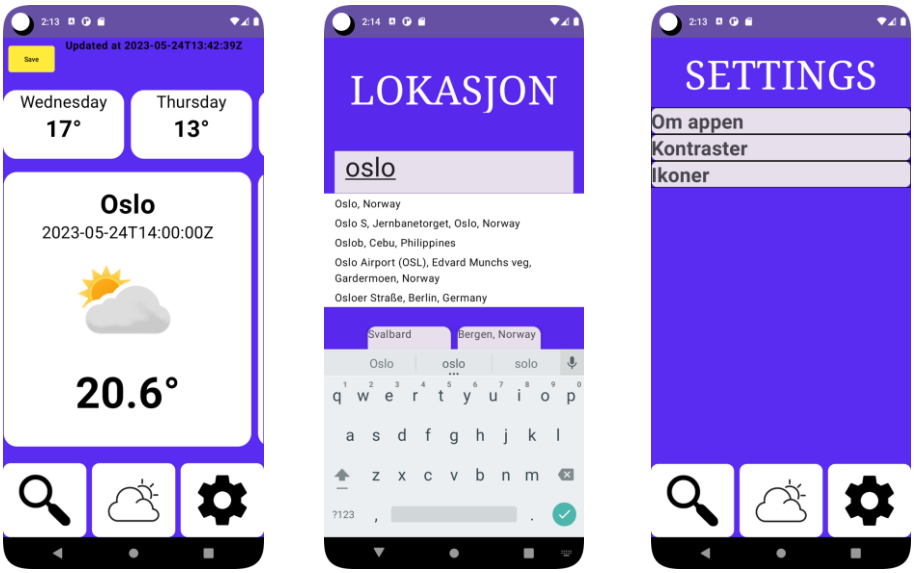
_____
Signatur

Vedlegg av bilder:

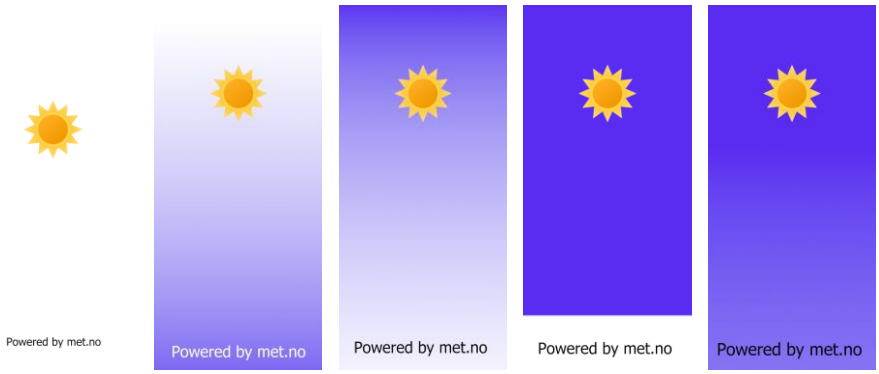
vedlegg 2 (skisse av første iterasjon av MainScreen med % størrelse for komponenter)



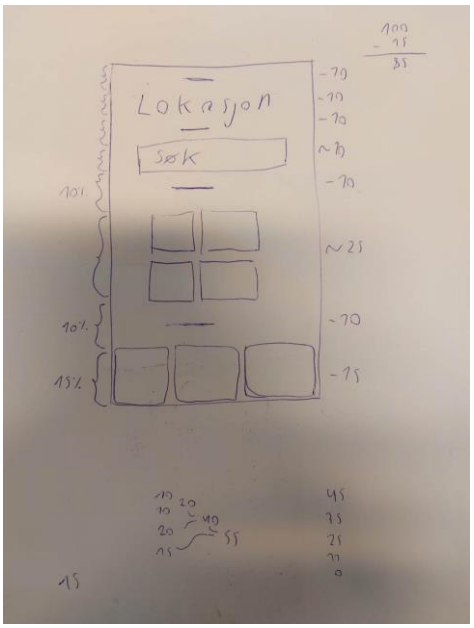
Vedlegg 3 (Applikasjonen under brukertesting)



Vedlegg 4 (design varianter av loadingScreen. bakgrunnsbilde nummer 3 midterste ble valgt)

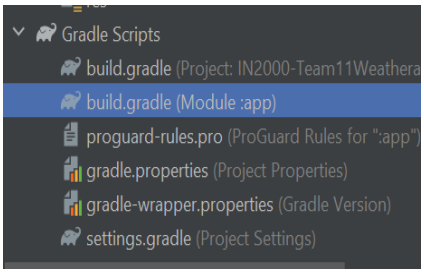


Vedlegg 5 (skisse av SearchScreen)

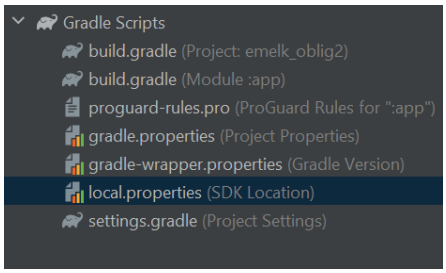


Vedlegg 6 (Problemer som oppsto med android studio for en i gruppent)

```
SDK location not found. Define a valid SDK location with an ANDROID_HOME environment
 variable or by setting the sdk.dir path in your project's local properties file at
 'C:\Users\emelk\AndroidStudioProjects\IN2000-Team11WeatherappWeakSight\local.properties'.
```



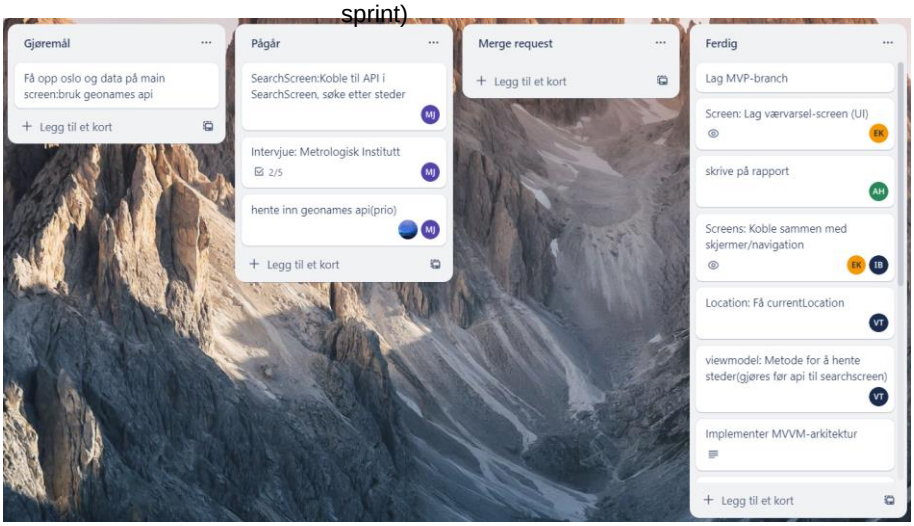
gradle scripts for brukeren med problem kjøre appen



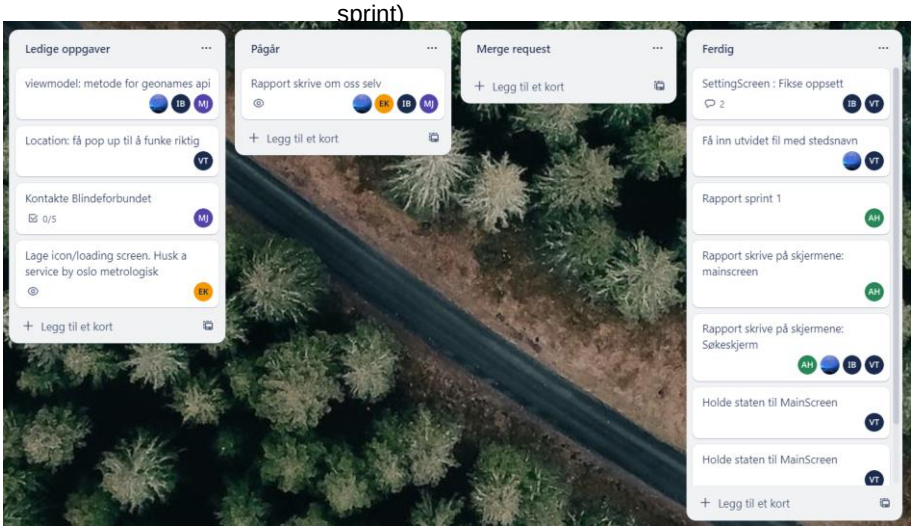
gradle scripts for en som kunne

Vedlegg av trello boards

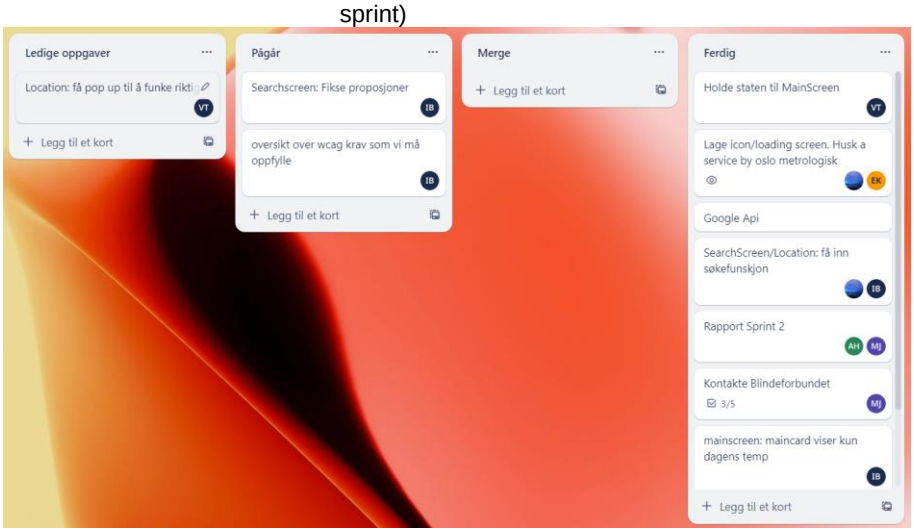
vedlegg 7 (første sprint)



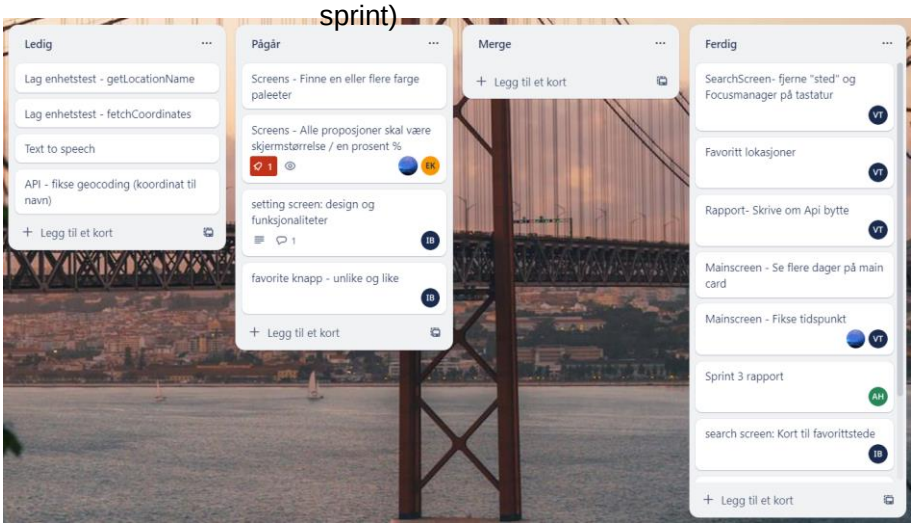
Vedlegg 8 (andre sprint)



Vedlegg 9 (tredje sprint)



Vedlegg 10 (fjerde sprint)



Vedlegg 11 (Kartverket API)

https://kartkatalog.geonorge.no/metadata/soeketjeneste-for-stedsnavn/d12de000-1a23-46b3-9192-3a1a98b2c994

https://ws.geonorge.no/stedsnavn/v1/

https://www.kartverket.no/api-og-data/stedsnavndata/brukarrettleiing-stadnamn-api